FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Planeamento de Trajetórias em Ambientes Agrícolas

Luís Carlos Feliz Santos

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Prof. Dr. Pedro Luís Cerqueira Gomes da Costa

Coorientador: Prof. Dr. José Luís Sousa de Magalhães Lima

17 de Julho de 2017

Resumo

O Vinho do Porto é um produto de exportação que tipicamente é produzido em vinhas instaladas ao longo do rio Douro. No entanto, estas vinhas são caraterizadas por estarem instaladas ao longo das encostas das montanhas, com altos níveis de pedregosidade e com reduzido espaço de manobra para as máquinas. A mecanização e robotização destas vinhas é um desafio. Um dos desafios impostos, pelos declives destas vinhas, aos operadores de máquinas agrícolas e aos robôs é a estimação e execução de trajetórias seguras. Neste contexto, pretende-se encontrar uma solução viável de planeamento de trajetórias seguras que considere o centro de massa do robô e a inclinação do terreno. A solução proposta nesta dissertação consiste na adaptação de um algoritmo de planeamento de trajetórias, o A*, que tem em conta quatro variáveis fundamentais: as coordenadas da localização do robô no mundo (x, y), a orientação (θ) e a altitude do terreno. Estas variáveis serão utilizadas para verificar a segurança do caminho para o robô. Com base nestas variáveis será verificada a postura do robô em cada ponto da trajetória e garantido que o posicionamento do centro de gravidade do robô não ultrapassa os limites de segurança. A esta solução deu-se o nome de AgrobPP ("Agrob Path Planning"), sendo que este algoritmo é competente no planeamento trajetórias seguras para robôs que operem em contexto de vinhas montanha. No decurso deste trabalho, também se dotou o AgrobPP de uma capacidade para replaneamento local de trajetórias seguras, caso durante a execução da trajetória, o robô detete um obstáculo. O AgrobPP, partindo de um conjunto de pontos fornecidos pelo A* secciona estes "waypoints"em subsecções que são descritas por trajetórias paramétricas, úteis para controlo preciso do robô e estimação da energia gasta pelo robô durante a execução da trajetória. Por fim, o AgrobPP ainda tem a funcionalidade de planear caminhos em modo "off-line", que serão posteriormente utilizados para deslocar o robô até um dos postos de carregamento localizados no terreno. Para escolher esse caminho, utiliza-se a informação da energia gasta pelo robô durante a execução da trajetória.

Abstract

Port wine is an export product which is typically located in the vineyards along Douro river. However, these vineyards are characterized for being along the slopes of the mountains, with high levels of stoniness and reduced space for machines. The mechanization and robotization of these vines is a challenge. One of the problems, caused by the slopes of these vineyards, to the operators of agricultural machinery and Robots is the estimation and execution of safe trajectories. In this context, we intend to find a safe path planning solution that considers the mass center of the robot and the slope of the terrain. The solution proposed in this dissertation consists of adapting a trajectory planning algorithm, A *, which takes into account four fundamental variables: the coordinates of the robot's location in the world (x, y), orientation (θ), and altitude . These variables will be used to verify the security of the path to the robot. Based on these variables we will verify the position of the gravity center in each point of the trajectory to assure that it doesn't exceed the safety limits. To this solution was given the name of AgrobPP ("Agrob Path Planning"), being that this algorithm is competent in the planning of safe trajectories for robots that operate in context of mountain vineyards. In the course of this work, the AgrobPP was also equipped with a capacity for local replanning of safe paths, if during the execution of the trajectory, the robot detects an obstacle. AgrobPP, divides a set of points provided by the A* into subsections that are described by parametrical curves, useful for precise control of the robot and estimation of the energy spent by the robot during the execution of the trajectory. AgrobPP still has the functionality of planning off-line paths, which will later be used to move the robot to one of the field located loading stations. To choose this path, the information of the energy spent by the robot during the execution of the trajectory is used.

Agradecimentos

Ao meu orientador, Prof. Dr. Pedro Costa e ao meu coorientador Prof. Dr. José Lima pelo apoio, disponibilidade e orientação prestados durante a realização desta dissertação.

Ao projeto ROMOVI pelos dados fornecidos para fins de desenvolvimento desta tese de mestrado. O ROMOVI é um projeto financiado pela comissão europeia no âmbito do programa Portugal 2020 no âmbito do acordo de subvenção número 17945 (ROMOVI), para a Investigação e Desenvolvimento cooperativo.

Ao Dr. Filipe Neves dos Santos, responsável pelo projeto ROMOVi, pela sua disponibilidade e grande contribuição para a realização desta dissertação.

Ao Dr. Héber Sobreira e ao Dr. Bruno Ferreira pela sua contribuição.

Aos meus Pais, Irmãs e restante família pelo apoio prestado não só ao longo da dissertação mas de todo o meu percurso académico.

A todos os meus amigos pelo companheirismo e ajuda prestados ao longo do meu percurso.

À minha namorada por todo o apoio, incentivo e ânimo prestado durante este percurso.

Obrigado.

Luís Santos

"In the twenty-first century, the robot will take the place which slave labor occupied in ancient civilization. '

Nikola Tesla

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ROMOVI	Robô Modular e Cooperativo para Vinhas de encosta
ROS	"Robot Operating System"
RRT	"Rapidly-exploring random tree"
AgrobPP	"Agrob Path Planning"

Capítulo 1

Introdução

1.1 Contexto

Desde há algumas décadas que se procura substituir o trabalho humano por máquinas autónomas, robustas e eficientes sendo que, para tal, a investigação na área da robótica tem um papel crucial. Tendo em conta que a População Mundial tem tendência a aumentar, atingir-se-á um ponto em que a produção agrícola pelos métodos atuais deixará de ser suficiente, sendo necessário tornar a produção cada vez mais eficiente e que ocupe espaços não explorados.

Neste sentido, uma das estratégias para otimizar a produção consiste em recorrer a robôs autónomos para auxílio nas atividades agrícolas. No entanto, as opções comerciais disponíveis neste momento nem sempre se ajustam e satisfazem as necessidades que se fazem sentir. Com efeito, embora se tenham vindo a desenvolver alguma soluções nas últimas décadas, a verdade é que, em situações em que as tarefas são muito específicas e de difícil aplicação, ou em culturas de difícil acesso, muito comuns em Portugal, como por exemplo, as vinhas de encosta, onde são produzidos alguns dos melhores vinhos a nível mundial, grande parte dos trabalhos ainda são feitos manualmente, como a poda e a própria vindima, pois não existem alternativas devido às particularidades do terreno.

Atendendo a esta conjuntura, o Centro de Robótica Industrial e Sistemas inteligentes, do INESC TEC, desde 2014 que delineou uma linha de investigação denominada de robótica para agricultura, denominada de AGROB[1], que se foca no desenvolvimento de soluções de robótica para aplicação ao contexto de vinha de montanha. Neste momento esta linha conta com 3 plataformas disponíveis: o AGROB V16, AGORB V15, AGROB V14, visíveis na Figura 1.1, sendo que, o AGROB V16 está a ser desenvolvido de forma modular para a execução de tarefas que vão desde a medição da variabilidade/sensorização/monitorização (em desenvolvimento) a tecnologias de atuação como a pulverização de precisão, poda e colheita (a desenvolver no futuro).[1] Em paralelo o projeto RoMoVi -Robot Modular e Cooperativo para Vinhas de encosta, pretende

[1] http://www.agrob.inesctec.pt

[1] PRÉMIO "O MELHOR DO PORTUGAL TECNOLÓGICO"ATRIBUÍDO A ROBÔ DO INESC TEC, https://https://www.overleaf.com/dashwww.inesctec.pt/crob/noticias-eventos/noticias/premio-o-melhor-do-portugal-tecnologico-atribuido-a-robo-do-inesc-tec

Figura 1.1: Plataformas robóticas da linha de investigação AGROB do CRIIS-INESCTEC

desenvolver máquinas de pequeno/médio porte totalmente autónomas e robustas para diversas tarefas agrícolas, sendo que um dos focos principais serão as vinhas de encosta. Este trabalho de dissertação é desenvolvido no contexto do projeto RoMoVi e linha de de investigação apelidada de AGROB.

Esta dissertação envolve-se,assim, no projeto RoMoVi, nomeadamente no contexto de planeamento de trajetórias seguras. Pretende-se portanto, garantir que as plataformas robóticas possam navegar em total segurança em contextos vinhas de encosta.

1.2 Motivação

Em Portugal, os terrenos vinícolas de maior valor têm a particularidade de se situarem em encostas de elevada inclinação e com alta pedregosidade, dificultando as tarefas dos robôs agrícolas, que, seja pela forte inclinação ou pela possível má aderência das rodas ao solo, correm o risco de sofrer acidentes que danifiquem o robô e o equipamento que este transporta. Pretende-se portanto, encontrar uma solução de planeamento de trajetórias seguras que considere o centro de massa do robô e inclinação do terreno, solução que será integrada no projeto "RoMoVi".

1.3 Objetivos

Tendo em conta a motivação descrita, o objetivo principal desta dissertação passa por desenvolver uma solução que aplicada a um algoritmo de planeamento de trajetórias funcione em terrenos de condições pouco favoráveis com forte inclinação. Pretende-se que no final desta dissertação se tenha um algoritmo de planeamento de trajetórias que considere a variação do centro de massa do robô, e que esse algoritmo esteja implementado e testado nos robôs disponíveis para este projeto, de modo a que estes robôs se possam deslocar em segurança ao longo de uma vinha de encosta.

Também se pretende que o algoritmo esteja preparado para replanear a trajetória caso surja um obstáculo inesperado a meio do seu percurso, ou seja, ter um método de evitar obstáculos locais.

Devido às grandes dimensões destes terrenos, nenhum dos robôs do projeto "RoMoVi"tem autonomia suficiente para percorrer toda a vinha, pelo que serão colocados postos de carregamento em várias zonas. O algoritmo desenvolvido nesta dissertação, poderá ser reaproveitado para gerar um caminho até um posto de carregamento que não terá que ser obrigatoriamente o mais próximo, mas sim aquele que exigir um menor custo energético ao robô.

A realização enquadra as seguintes etapas:

- Estudar e escolher um algoritmo de planeamento de trajetória adequado
- Adaptar o algoritmo para o objetivo desta dissertação
- Testar este algoritmo numa vinha em ambiente simulado
- Acrescentar a capacidade de replanear parte da trajetória gerada tendo em conta os obstáculos locais
- Reaproveitar o algoritmo gerado para planear trajetórias até aos postos de carregamento

1.4 Estrutura da Dissertação

Para além da introdução, esta dissertação contém mais 5 capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, faz-se um pequeno estudo ao algoritmo A*, e aborda-se o trabalho de adaptação às ferramentas de trabalho. No capítulo 4 começa por se fazer um estudo a um algoritmo A* já existente, explicando de seguida como se procedeu para que este algoritmo pudesse considerar a variação do centro de massa do robô tendo a demonstração de alguns resultados no fim deste capítulo. É também abordado aqui o método de desvio de obstáculos locais. No capítulo 5 descreve-se como será abordado o planeamento "off-line"de trajetórias para os postos de carregamento, tendo também os resultados expostos no fim do capítulo. No capítulo 6 tiram-se conclusões do trabalho feito e dão-se sugestões de trabalho futuro

Capítulo 2

Revisão Bibliográfica

Neste capítulo serão abordados conceitos teóricos fundamentais ao desenvolvimento da dissertação. Serão estudados vários algoritmos de planeamento de trajetórias e as plataformas robóticas disponíveis, de modo a poder analisar as vantagens e desvantagens de cada um, para que seja feita uma escolha adequada do algoritmo a desenvolver.

2.1 Planeamento de trajetórias

A navegação de um robô móvel envolve o seguinte conjunto de tarefas, ilustrado na Figura 2.1: perceção do ambiente, localização e construção do mapa, planeamento de trajetória e controlo do movimento. Sendo esta dissertação uma tarefa de planeamento de trajetórias, tem como objetivo encontrar um caminho entre dois pontos específicos, evitando obstáculos e otimizando critérios como tempo, distância e energia. Existe um largo número de abordagens, sendo que estas se dividem em duas categorias: “Off-line path planing” e “On-line path planning” em que a primeira categoria é utilizada em ambientes onde o robô tem acesso prévio a informação sobre qualquer tipo de obstáculo estático e trajetórias de obstáculos moveis. Esta é uma abordagem que também é conhecida por "global path planning". Quando a informação sobre os obstáculos não existe ou está incompleta, o robô completa-a através de sensores à medida que se desloca no ambiente, sendo então utilizada a segunda categoria de planeamento, "On-line path planning", começando por funcionar da mesma forma que o modo “off-line”, mas com a capacidade de alterar o percurso se forem detetadas mudanças nos obstáculos do ambiente[1].

À partida, o robô terá, assim, acesso a um mapa da vinha de encosta com informação sobre todos os obstáculos, contudo, não se excluí a possibilidade da aparição de um obstáculo inesperado, portanto, o algoritmo a ser desenvolvido deverá ter a capacidade de replanear a trajetória. Nos próximos sub-capítulos serão abordados diferentes algoritmos de planeamento de trajetória, que são baseadas em 3 conceitos: O campo de potencial (“Potential field planners”), a pesquisa em árvore (“Treebased planner (RRT)”) e mapas de grelhas.

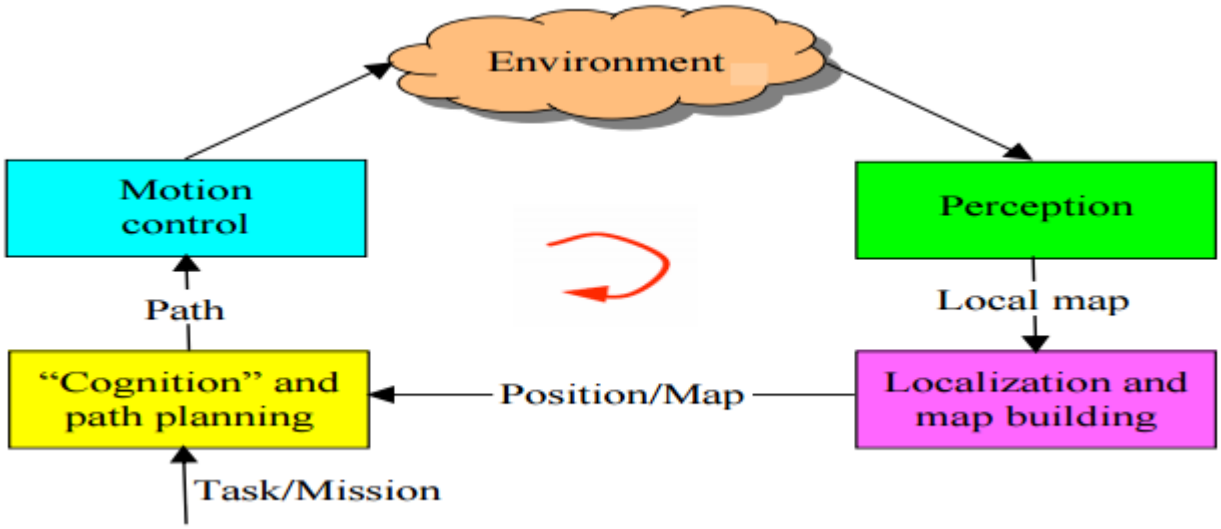

Figura 2.1: Tarefas para navegação de um robô móvel [1]

2.1.1 Campo de Potencial

Num campo de potencial o robô comporta-se como uma partícula imersa num campo potencial, em que os obstáculos representam potenciais de repulsão e o ponto de destino cria um potencial de atração, visível na Figura 2.2. Este algoritmo é utilizado em ambientes estacionários e não é ótimo, pois nem sempre encontra uma solução, sendo um dos seus principais problemas a criação de mínimos locais que acontecem por exemplo, quando o(s) obstáculo(s) e o destino estão alinhados como visível na Figura 2.3, em que há a possibilidade do potencial de repulsão ser maior ou igual que o potencial de atração, sendo impossibilitada a chegada do robô ao ponto de destino [11].

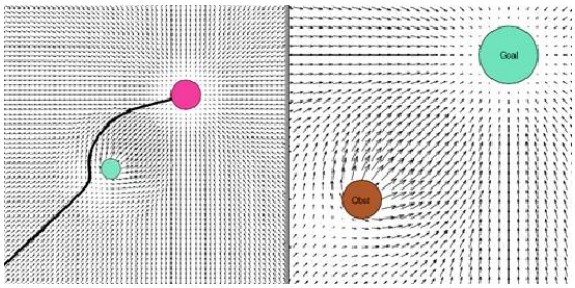

Figura 2.2: Representação dos potenciais de atração e repulsão [2]

Apesar desta abordagem ser utilizada em ambientes estáticos, existem algoritmos modificados com soluções que funcionam em ambientes dinâmicos, que aproveitam o facto do ambiente não ser estático para resolver o problema dos mínimos locais. Uma opção consiste em esperar que um eventual movimento do obstáculo ou do ponto de destino acabe com o mínimo local. Contudo, se isto não resolver o problema, tenta-se acabar com o mínimo local por outros meios, como por

exemplo, fazer o robô contornar o obstáculo com um algoritmo para seguir paredes, ou efetuar pequenos movimentos aleatórios com o robô até que o mínimo local desapareça. [3]

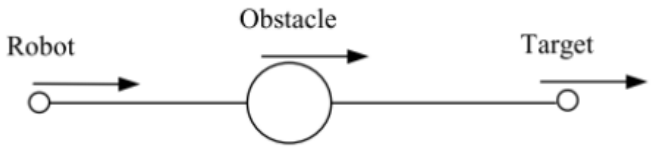

Figura 2.3: Problema de minimo local [3]

2.1.2 Pesquisa em Árvore (RRT)

Na pesquisa em árvore ("rapidly exploring random tree- RRT) o caminho é explorado de forma aleatória, que planeia a trajetória entre dois pontos tendo em conta a dinâmica do robô, sendo especialmente desenhado para veículos não holonómicos, isto é, veículos com restrições no seu movimento. Estes algoritmos têm a vantagem de ser simples e de fácil compreensão, contudo têm o problema de nem sempre encontrarem o melhor caminho possível, gerando trajetos com muitas curvas que podem dar origem a erros de odometria [4] [12].

Samuel Rodriguez sugere o seu algoritmo RRT baseado em informação dos obstáculos para gerar um caminho com menos curvas, podendo verificar o resultado na Figura 2.4.

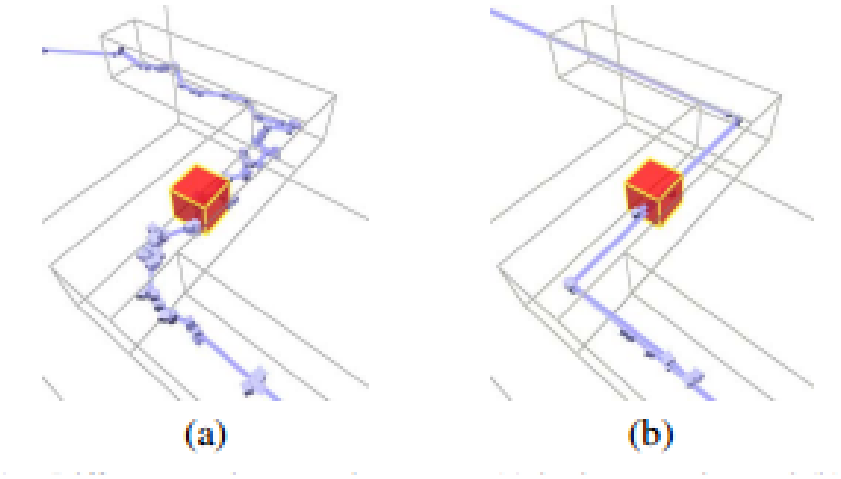

Figura 2.4: (a) RRT básico e (b) RRT com informação dos obstáculos [4]

2.1.3 Mapas de grelha

Os mapas de grelhas em si não representam algoritmos de planeamento de trajetórias, são apenas mapas com diversas utilidades, sendo uma delas o uso em algoritmo de planeamento de trajetória. Este mapas consistem em dividir o mapa em células, em que cada célula tem informação

sobre a sua disponibilidade, procurando a melhor solução considerando o movimento restrito às células livres.

A decomposição em células pode ser feita com recurso a:

- Decomposição em células exatas
- Decomposição em células aproximadas

Na decomposição em células exatas, o mapa representa exatamente o mundo real, isto é, as células representam ou espaços livres ou espaços ocupados. Já nas células aproximadas, tem-se acesso a mais informação, cada célula pode estar livre, ocupada ou parcialmente ocupada. Geralmente, as células são representadas com um quadrado de dimensão variável, e podem ser dividas como células fixas (Figura 2.5), tendo sempre o mesmo tamanho pré-definido, ou em "quadtree"(Figura 2.6), onde se divide recursivamente o espaço em quatro células iguais e sempre que uma célula tiver um obstáculo, volta a dividir-se por quatro até atingir um limite definido [2].

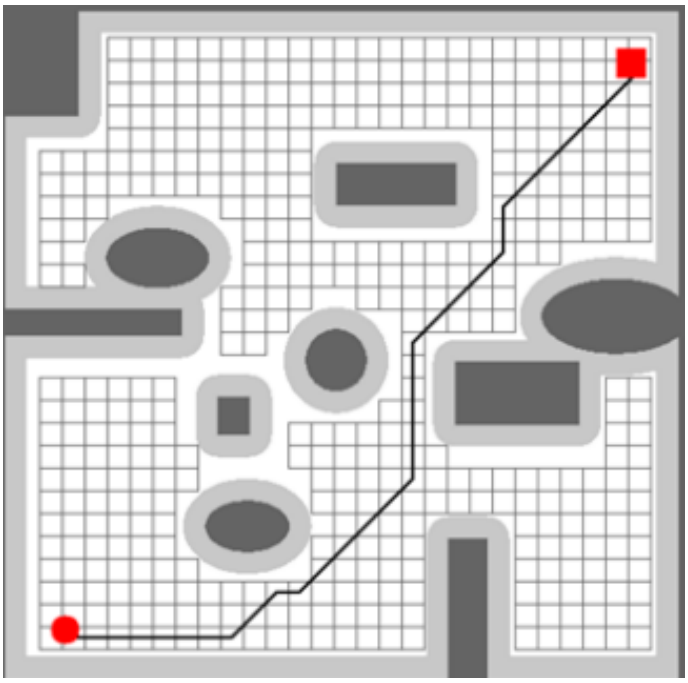

Figura 2.5: Decomposição em células fixas [2]

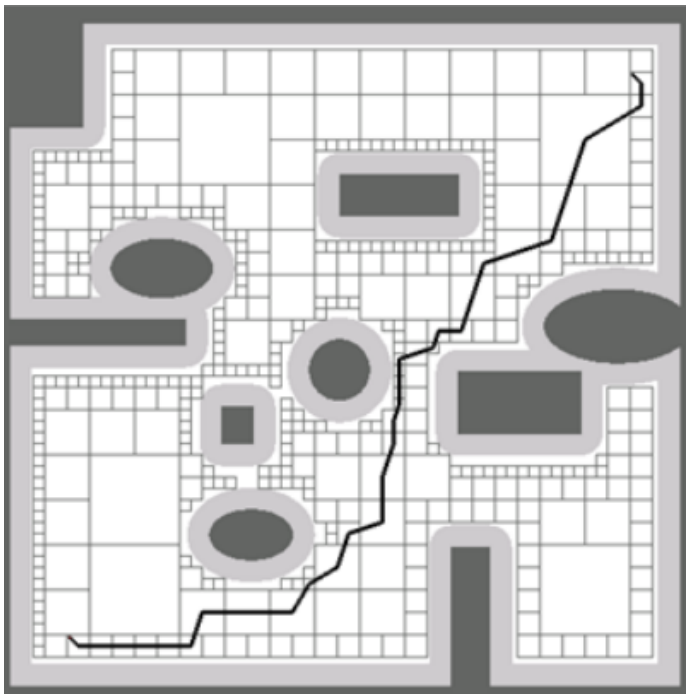

Figura 2.6: Decomposição em "quadtree" [2]

Com este tipo de mapas, as abordagens mais conhecidas são os algoritmos o “Dijkstra” e o A*, dois algoritmos similares sendo que o A* se destaca por ter um tempo de processamento menor e espaço de procura menor que o "Dijkstra", algo que se verifica no exemplo da Figura 2.7,sendo que, habitualmente, o A* é a escolha mais adequada para grande parte dos problemas.

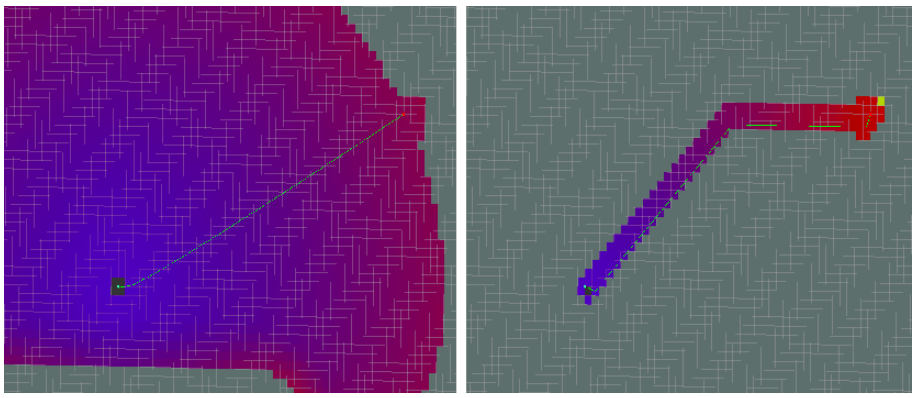

Figura 2.7: Dijkstra (à esquerda) e A* (à direita) [5]

2.1.3.1 Algoritmo A*

Tal como já foi mencionado, o algoritmo A* é ótimo, ou seja, encontra sempre o melhor caminho entre dois pontos, que pode ser o caminho de menor distância, menor gasto energético, menor tempo de execução, entre outros. Mas tem a contrapartida de ser computacionalmente

pesado, sendo necessário ter em consideração vários aspetos na sua implementação tais como o poder de cálculo do processador, a limitação de memória e o tipo de heurística a utilizar. Irão ser apresentadas abordagens que utilizam versões modificadas do A* para melhorar certos parâmetros do algoritmo, sendo que, três destas abordagens são trabalhos que foram desenvolvidos dentro da Faculdade de Engenharia da Universidade do Porto:

- Tempo de Processamento: Takayuki Goto propõe a implementação do algoritmo A* com heurística inteligente, calculando uma heurística quase ótima como uma função constante. Este artigo propõe uma abordagem que varia entre o algoritmo A* e o A (equivalente ao A* com diferença na qualidade da heurística), sendo que, se a heurística for admissível, o A* determina o caminho ótimo, caso contrário é usado o algoritmo que determina um caminho mais longo, mas mesmo neste caso o caminho calculado não é mau. Na prática, esta abordagem poupa tempo de cálculo, mas a qualidade do trajeto determinado é invariante [13].

- Obstáculos não estáticos: Este algoritmo desenvolvido na Faculdade de Engenharia da Universidade do Porto, com a participação do Prof. Pedro Costa, sugere um algoritmo A* modificado com o objetivo de o otimizar para obstáculos não estáticos, tendo apresentado um algoritmo que, apesar de ter um tempo de processamento maior, atinge o ponto de destino mais rapidamente, tendo o resultado ilustrado na Figura 2.8 [6]. Uma outra abordagem desenvolvida na Faculdade de Engenharia da Universidade do Porto, apresenta um conjunto de modificações que podem ser aplicadas a qualquer algoritmo da família A*. São apresentadas cinco modificações para que o robô possa atingir o ponto de destino mais rápido que os algoritmos tradicionais, assim como evitar obstáculos que se movam tão rápido (ou até mesmo mais rápido) que o robô, sendo que as modificações são as seguintes: Distância do obstáculo, “slack”, direção, tempo de processamento e orientação do ponto de destino [14].

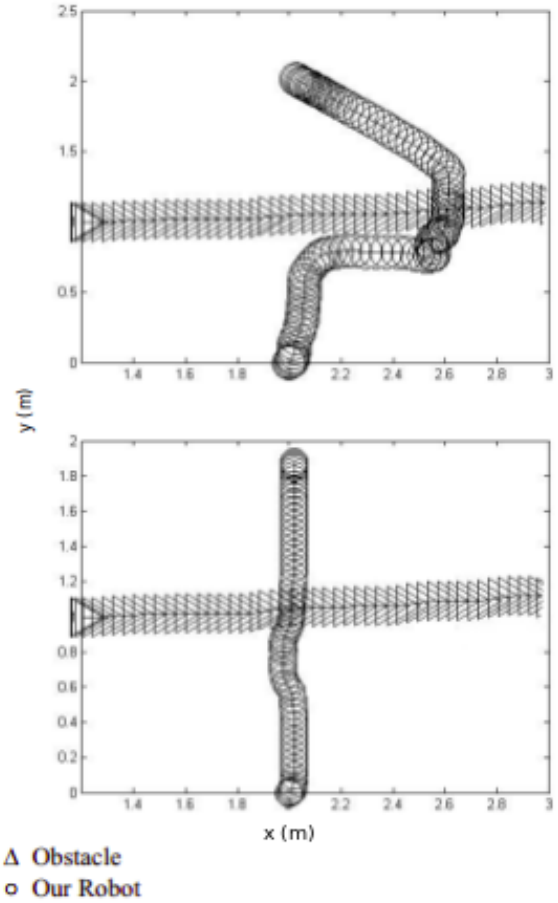

Figura 2.8: a) A* Tradicional (em cima) b) A* modificado (em baixo) [6]

- Orientação do robô: grande parte dos algoritmos analisados apenas consideram duas dimensões que representam a posição do robô num espaço bidimensional. Esta versão do algoritmo A* desenvolvida na Faculdade de Engenharia da Universidade do Porto, introduz uma terceira variável ao algoritmo, a orientação do robô, sendo que um gerado com este A*, tem em conta a orientação do ponto de origem, do ponto de destino e do robô ao longo do percurso, limitando o algoritmo às características do robô. Neste caso, o algoritmo foi desenvolvido no projeto "CARLoS"[1], para um robô móvel com um manipulador integrado. Para tal, o mapa de grelhas tradicionalmente utilizado no A*, é multiplicado para diferentes orientações do robô, e preenchido de forma diferente em cada orientação, utilizando o mapa correspondente à orientação atual do robô [7]. Na Figura 2.9 tem-se o resultado deste A* comparado com um algoritmo Dijkstra tradicional, onde, a azul está representada uma trajetória gerada pelo A* com orientação, e a vermelho o resultado de um planeamento com o "Dijkstra", em quem é visível que a trajetória gerada não considera qualquer restrição do robô enquanto que o A* gera um caminho suave e considera a orientação da plataforma.

[1] "CooperAtive Robot for Large Spaces manufacturing", https://www.inesctec.pt/crob/projetos/projectos-em-destaque/carlos/

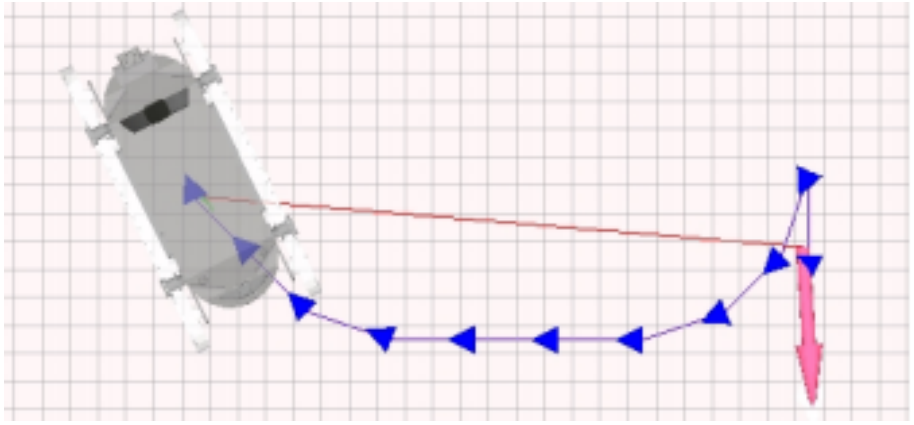

Figura 2.9: Dijkstra vs A* com Orientação [7]

- Terrenos irregulares: sendo o principal problema desta dissertação a irregularidade do terreno, procurou-se pesquisar soluções já desenvolvidas nesta área. Contudo, é um tema pouco abordado sendo que, o caso encontrado mais similar a esta dissertação, foi um algoritmo da família A*, o D*, modificado para navegar em terrenos irregulares com altitude variável, tendo o resultado representado na Figura 2.10 [8].

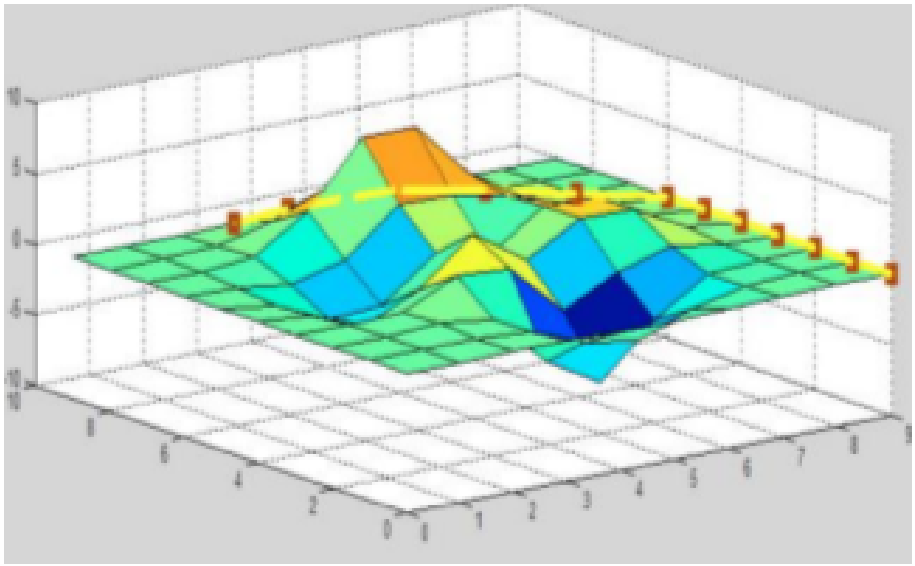

Figura 2.10: Algoritmo D* modificado [8]

2.2 Projetos em Vinhas de encosta

Apesar de este tema não ser muito comum, foi encontrado um projeto feito para vinhas de encosta que consiste em aplicar uma operação de esqueletização a cada socalco da vinha até este ficar reduzido a uma linha de trajetória (Figura 2.11), facilitando a navegação autónoma de robôs

móveis, porém, esta proposta não entra em consideração com a inclinação da vinha que o robô precisa de vencer para se deslocar entre socalcos [9].

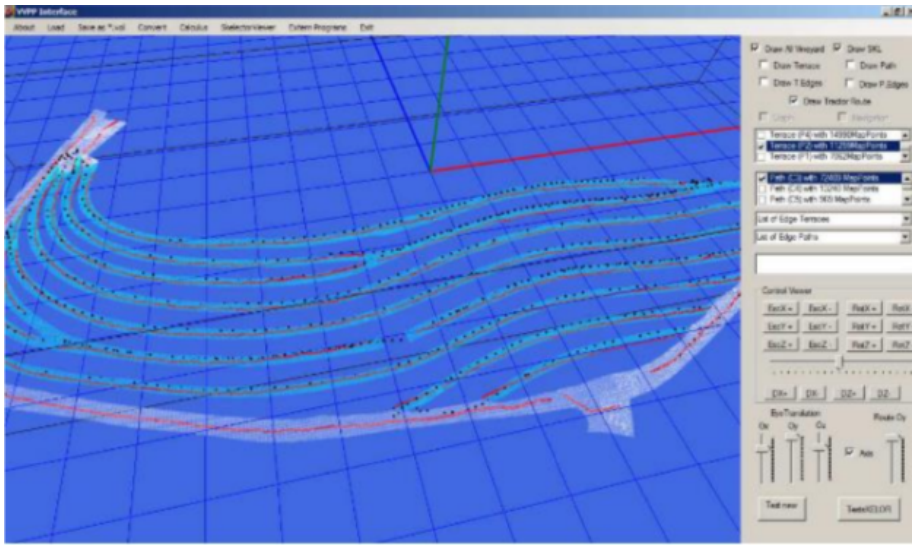

Figura 2.11: Trajetória utilizando a esqueletização [9]

2.3 Plataformas robóticas

Como já foi mencionado anteriormente, neste momento existem duas plataformas robóticas disponíveis: o AGROB14 e o AGROB16, representados na Figura 1.1. Foi construído um modelo simulado do AGROB14 com recurso ao simulador Gazebo para poder testar um robô de tração "Ackerman"num ambiente externo usando ROS, tendo também disponível um modelo simulado de uma vinha de encosta, como mostram as Figuras 2.12 e 2.13 [15]. O autor disponibiliza ainda uma versão do robô simulado para o Rviz, um programa muito útil para tarefas de visualização como: mapas, trajetos gerados, informação sobre leitura de sensores, entre outros.

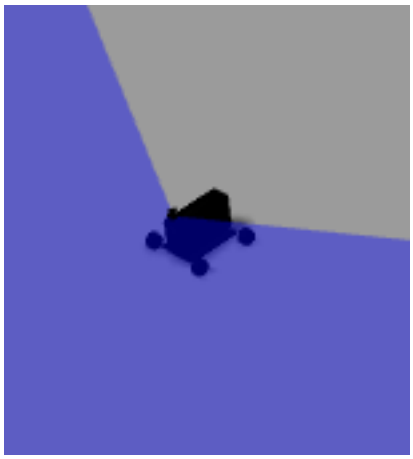

Figura 2.12: Versão simulada do AGROB14

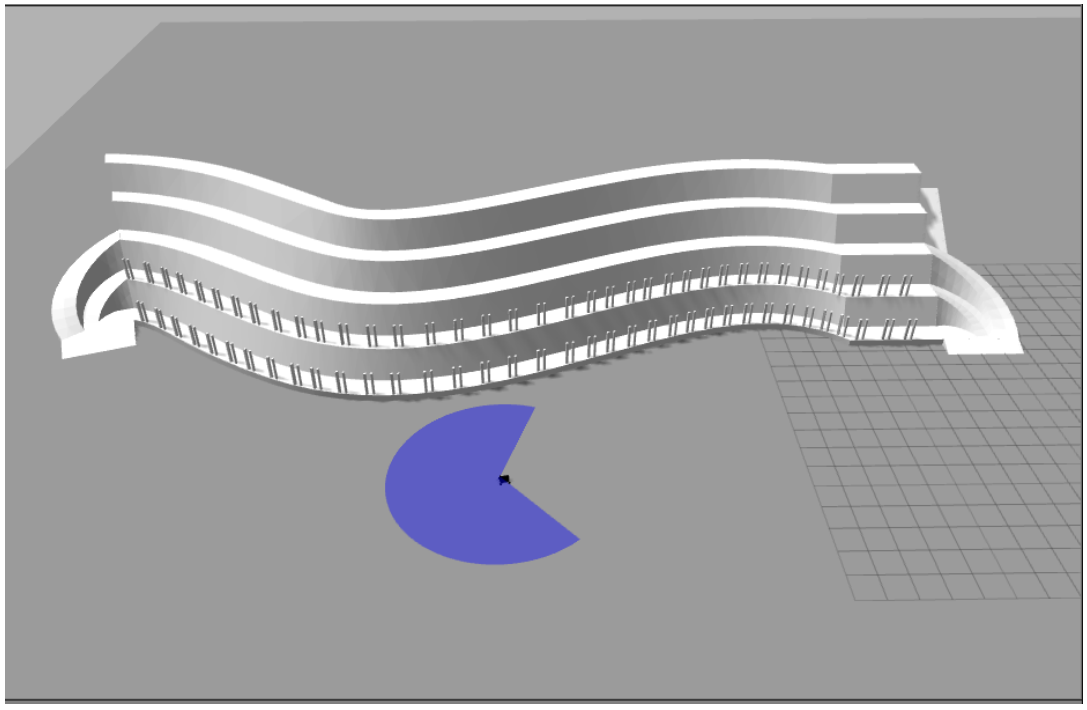

Figura 2.13: Modelo 3D de uma vinha de encosta

2.4 Estimação da Energia Consumida

Embora o objetivo primordial desta dissertação não consista na minimização de energia gasta pelo robô durante o percurso, será útil, ainda assim, estimar a energia que o trajeto gerado irá consumir, já que foi definido que o algoritmo desenvolvido irá ser utilizado para determinar um caminho até um posto de carregamento, em que o posto de carregamento mais próximo poderá ser preterido por outro que esteja mais longe mas tenha um custo energético menor. Uma forma

de calcular a energia consiste em transformar o caminho gerado que normalmente é constituído por "waypoints"em curvas, suavizando deste modo a trajetória o que também é útil para um futuro controlo do robô.

Uma solução proposta consiste em formar uma curva cúbica de Bézier (equação 2.1) entre cada dois pontos da trajetória, descrevendo toda a trajetória com este tipo de curvas. Em que (x_i, y_i), *e* (x_{i-1}, y_{i-1}) representam dois pontos da trajetória, (x_a, ya) *e* (x_b, yb) são ponto intermédios calculados com base nos pontos da trajetória e u é um parâmetro que varia entre 0 e 1.

$$\begin{cases} x(u) &= (1-u)^3 x_{i-1} + 3u(1-u)^2 x_a + 3u^2(1-u)x_b + u^3 x_i \\ y(u) &= (1-u)^3 y_{i-1} + 3u(1-u)^2 y_a + 3u^2(1-u)y_b + u^3 y_i \end{cases} \tag{2.1}$$

De seguida, com a fórmula do consumo energético visível na equação 2.2, é possível calcular a energia gasta pelo robô entre cada dois "waypoints"$(i-1, i)$, tendo em conta o atrito (μ), a velocidade (v), a aceleração e a energia perdida a transformar energia elétrica em mecânica (ks). Tanto a velocidade como aceleração são derivadas das curvas de Bezier que descrevem a trajetória [16].

$$E_{i-1,i} = \int_{T_{i-1,i}} ks\,dt + \int_{T_{i-1,i}} \mu mg v_{i-i,i}(t)\,dt + \int_{T_{i-1,i}} d(\frac{1}{2}.mv_{i-1,i}(t)^2)\,dt \tag{2.2}$$

2.5 Conclusões

Antes da escolha de um algoritmo adequado, terá que se ter em conta a plataforma robótica que irá ser utilizada, tendo já visto na Figura 1.1 que há duas plataformas disponíveis, uma com configuração diferencial, o Agrob16, e outra com configuração tipo carro ("Ackerman"), o Agrob14. Optou-se por considerar o robô com configuração "Ackerman", visto já existir um modelo simulado do mesmo, contudo o algoritmo desenvolvido deverá funcionar para os dois robôs, apenas se altera forma de controlo de cada um.

Tendo em conta os algoritmos estudados, verifica-se que os mapas de grelhas fazem parte de soluções mais recentes que os restantes e, contêm um algoritmo em particular, o A*, que é ótimo. Ora, para a escolha de um método é aconselhável ter em conta três parâmetros:

- O tipo de otimização pretendida (distância, tempo, energia, entre outros)
- A complexidade computacional
- Se o método é completo

Para esta dissertação pretende-se otimizar a distância percorrida, pelo que para este ponto há vários algoritmos possíveis, contudo se mais tarde for pretendido otimizar variáveis como tempo ou energia, é possível fazê-lo modificando o algoritmo, algo que é relativamente simples se estiver a ser utilizado um A*. Tal como já foi indicado o A* é um método completo, restando apenas avaliar a complexidade computacional, que nesta opção se pode tornar demasiado elevada, impossibilitando a sua implementação. Apesar desta contrapartida, o A* poderá ser uma boa escolha para resolver o problema.

Restringir o algoritmo à orientação do robô é muito útil para esta situação em que se pretende que o centro de gravidade do robô esteja situado numa zona segura, uma vez que no mesmo ponto o robô poderá estar ou não em segurança dependendo da sua orientação. Se não se considerar esta restrição, haverá o risco de certas zonas perigosas serem consideradas seguras e vice-versa. Pretende-se também que no futuro as plataformas do projeto "ROMOVi"sejam equipadas com manipuladores para efetuar tarefas de poda e colheita. Ora, o projeto "CARLoS", apesar de ter sido desenvolvido para um fim completamente diferente, contempla um robô equipado com um manipulador, que atualmente tem como solução de planeamento de trajetórias um algoritmo A* modificado para contabilizar a restrição imposta pela orientação do robô, uma vez que com o manipulador, o robô terá que atingir o destino com uma certa orientação. Apesar de, atualmente, os objetivos serem diferentes, pretende-se também que nesta dissertação se implemente um algoritmo que tenha em conta a orientação do robô. Optou-se, então, por utilizar o algoritmo já desenvolvido no projeto "CARLoS"como base, sendo necessário acrescentar-lhe a restrição da altitude do terreno e do centro de gravidade do robô.

Assim sendo, esta dissertação irá apresentar uma solução para a navegação em vinhas de encosta, considerando um algoritmo A* restringido à orientação do robô e à posição do seu centro de gravidade.

Capítulo 3

Neste capítulo apresenta-se um pequeno estudo do algoritmo A* tradicional e a sua implementação no robô AGROB14 demonstrando o trabalho que foi feito numa fase inicial para adaptação ao algoritmo e às ferramentas de trabalho como o ROS, o Gazebo e o Rviz.

Resumidamente, foi implementado um algoritmo A* tradicional e fez-se um controlador simples para o robô seguir o caminho gerado. Só depois desta fase de adaptação é que se começou a trabalhar com o algoritmo restringido à orientação do robô .

3.1 Algoritmo A*

Já se abordaram alguns aspetos deste algoritmo no capítulo anterior, como o facto de utilizar mapas de grelha, e de ser um método ótimo, completo e de complexidade que pode ser elevada, dependendo da heurística utilizada, mesmo assim é o algoritmo mais eficiente dentro dos que utilizam heurística (Dijkstra e algoritmo guloso) [2]. Num A*, a função de custo que determina a ordem de pesquisa dos nós é dada por

$$f(n) = g(n) + h(n) \tag{3.1}$$

onde g(n) representa o custo atual do caminho desde o nó de início até ao nó n, e h(n) é uma função de heurística que estima o custo do nó atual n até ao nó de destino.

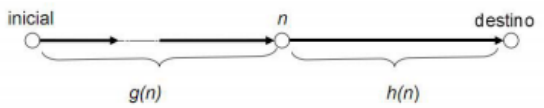

Figura 3.1: Função custo de um A* [2]

O algoritmo A* foi descrito inicialmente da seguinte forma: (Retirado de [2])

- "1) Começa-se com o nó inicial, calcula-se o seu f(n inicial) e coloca-se esse nó numa lista de nós designada por lista aberta. Os procedimentos seguintes são repetidos até se chegar ao nó de destino ou até a lista aberta ficar vazia, o que implica que não exista solução.

- 2) Escolhe-se o melhor nó n (aquele que tem a função de custo f(n) mais baixa) que se encontra na lista aberta. Este nó passa a ser considerado já processado, deixa de existir na lista aberta e passa a existir numa lista designada por lista fechada que contém todos os nós já processados.

- 3) Finalmente, avaliam-se todos os nós adjacentes (nós que estão diretamente ligados ao nó n), que podem ser um dos seguintes casos:

  - a) Serem inseridos na lista aberta se ainda lá não existirem e não pertencerem à lista fechada, com a informação do seu f(ni) e do seu antecessor, designado por Pai, ou seja o nó procedente de onde veio até chegar a ele.

  - b) Caso existam na lista aberta, vai-se determinar se a função f é menor que o valor da função f que estava anteriormente associada ao nó, isto é, verificar se seguindo este percurso a função custo é menor. Sendo este o caso, modifica-se o Pai do nó e o valor de f.

  - c) Se pertencer à lista fechada verifica-se se a função f é menor por este caminho do que quando esse nó foi processado. Se assim for, esse nó passa a pertencer outra vez à lista aberta.

  - d) A trajetória resultante é encontrada começando com o nó final e a partir dele vê-se qual é o seu Pai. Em seguida verifica-se nesse nó o seu Pai e assim sucessivamente até se chegar ao nó inicial."

Se a heurística for consistente o passo 3c) pode ser excluído, o que torna o algoritmo mais eficiente. Na Figura 3.2 encontra-se descrito o pseudocódigo algoritmo A* que adota a seguinte terminologia:

"**O** - A lista aberta contém os nós que ainda não foram selecionados e que podem vir a ser escolhidos.

C - A lista fechada contém os nós já processados, isto é, nós que já saíram da lista aberta, uma vez que já foram analisados.

Star(n) - Representa o conjunto de nós adjacentes do nó n, isto é, todos os nós que estão ligados ao nó n.

c(n1,n2) - É o custo de passar do nó n1 para nó n2

f(n) = g(n) + h(n) é a função custo do caminho mais curto desde o nó de início até ao nó final em que passa pelo nó n

g(n) - É o custo atual desde o início até ao nó n.

h(n) - É a função heurística de custo que faz a estimativa de ir do nó n até ao nó final."

1. Adicionar o nó de partida a *O*
2. Repetir
   - 2.1. Escolher o n_{melhor} (nó melhor) de *O* tal que $f(n_{melhor}) \leq f(n)$, $\forall n \in O$
   - 2.2. Remover o n_{melhor} de *O* e adicionar a *C*
   - 2.3. Se n_{melhor} = nó final , terminar o algoritmo
   - 2.4. Para todos n $\in$ Star(n_{melhor}) fazer o seguinte:
     - 2.4.1. Se ($n \notin O$) e ($n \notin C$) então adiciona o nó n a *O*
     - 2.4.2. Se ($n \in O$) então se $g(n_{melhor}) + c(n_{melhor}, n) < g(n)$ então alterar o pai do nó x para n_{melhor}
3. Até que O esteja vazio

Figura 3.2: Pseudocódigo do A* [2]

3.1.1 Propriedades

As seguintes propriedades, apresentadas em [2] mostram que o algoritmo A* é fiável e eficaz, sendo que o facto de ser completo é uma propriedade que já foi mencionada várias vezes.

- **Admissível** - O algoritmo é admissível se a heurística o for. Para tal, a função de heurística nunca pode sobrestimar o custo de chegar ao destino. Portanto, h(n) terá que ser sempre menor ou igual que o menor custo do nó n até ao destino.

$$h(n) \leq h_m(n)$$

- **Consistente** - Uma função h é consistente se para todos os nós em que exista ligação entre eles, a estimativa de chegar ao destino é sempre menor ou igual ao custo de chegar ao nó adjacente, mais a estimativa desse nó ao destino.

- "**Complexidade** - O tempo de execução do algoritmo A* depende da heurística, principalmente da qualidade da função heurística. Na pior heurística possível (h(n)=0) verifica-se que a complexidade é exponencial $O(2^n)$, no melhor cenário $h(n) = h_m(n)$ a complexidade é linear O(n)."

3.1.2 Heurísticas

Algumas das heurísticas sugeridas em [2] são:

- **Distancia Manhattan** - Só é possível mover-se em duas direções:

$$h(n) = D.((n.x - destino.x) + (n.y - destino.y))$$

- **Distancia euclidiana** - neste caso, pode mover-se em qualquer direção:

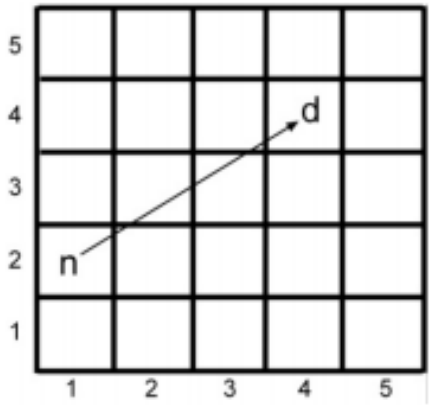

Figura 3.3: Heurística com distancia euclidiana [2]

3.2 Implementação do Algoritmo A* tradicional

Como o objetivo desta dissertação consiste em modificar um algoritmo A* já implementado, não havia necessidade de construir um de raiz, pelo que, nesta fase foi implementado um A* simples com recurso a uma biblioteca disponível online.

Uma vez que o principal objetivo desta implementação consistia na adaptação ao ROS, utilizou-se uma biblioteca disponível em [17]. Foi aproveitado o código fonte que dizia respeito a este algoritmo intitulado pelo autor como "Relaxed A*", que ainda está presente no código atual como uma biblioteca.

3.2.1 Mapa de ocupação

Como já foi mencionado, o A* pesquisa pelo melhor caminho através de mapas de grelhas, pelo que, a primeira tarefa consiste em fornecer esse mapa ao programa. Uma vez que se está a trabalhar em ROS, utilizou-se a ferramenta "map server", que lê com alguns parâmetros uma imagem e a transforma num mapa de ocupação, indicando em cada célula se está livre, ocupada, ou se desconhece o seu estado, sendo que esta informação é publicada por um tópico em ROS, que será subscrito pelo programa que contém o algoritmo. Tem-se na Figura 3.4 a imagem utilizada para a construção deste mapa que corresponde a um modelo 2D da vinha disponível no simulador Gazebo, e na Figura 3.5 a visualização de todas as células do mapa de ocupação com recurso ao Rviz, em que a cor branca representa um célula livre e a cor preta uma célula ocupada.

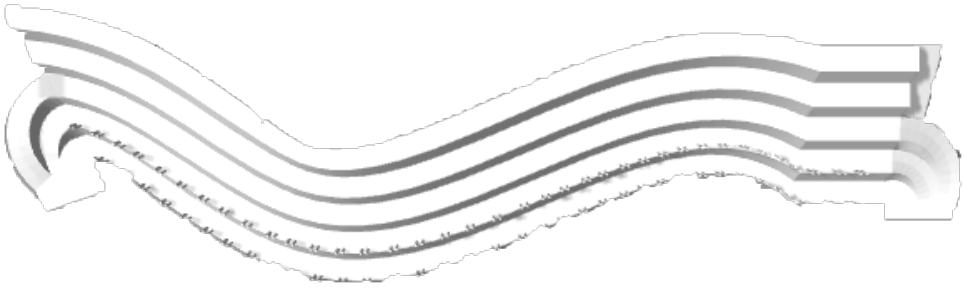

Figura 3.4: Mapa 2D de uma vinha de encosta

Figura 3.5: Mapa de Ocupação da vinha de encosta

3.2.2 "Relaxed"A*

Utilizou-se então a biblioteca de funções disponível em [17] para testar este algoritmo, sendo uma das contrapartidas detetadas o facto de o mapa utilizado por este algoritmo estar no formato "costmap 2d"do ROS , diferente do mapa fornecido pelo "map server"também do ROS. A diferença entre estes dois formatos prende-se na informação que cada um pode conter sendo que, o primeiro corresponde ao mapa dividido em células exatas, isto é, a célula ou está ocupada ou está livre, qualquer outro estado é interpretado como desconhecido. Já no "costmap2d", é possível fazer uma decomposição em células aproximadas, indicando se a célula está parcialmente ocupada. Foi então adicionada uma função ao código para converter de um formato para o outro assim que o "map server"enviar o mapa de ocupação. O resultado desta implementação está demonstrado na Figura 3.6, sendo claramente visível que é um algoritmo básico que contorna os obstáculos de forma tangencial sem restrições impostas por características do robô e funciona apenas num ambiente de duas dimensões.

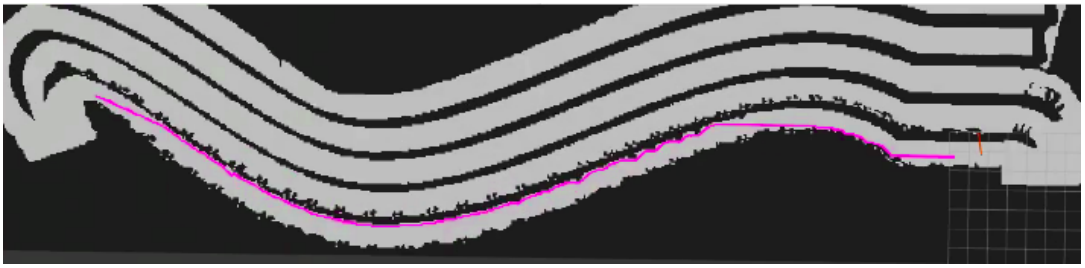

Figura 3.6: Caminho gerado pelo algoritmo "Relaxed"A*

3.2.3 Controlador

Este controlador foi desenvolvido para adaptação ao controlo de um veículo utilizando ROS, sendo apenas um simples controlador PI, apenas para que o robô se deslocasse em frente e fizesse pequenas correções à sua orientação de acordo com a trajetória gerada pelo A*. Apesar de ter sido feita uma simulação, não foi possível colocar o robô no modelo 3D da vinha uma vez que este algoritmo gerava caminhos muito próximos dos obstáculos. O veículo controlado foi então a versão simulada do robô AGROB14 que subscreve um tópico em ROS de velocidade e orientação, sendo possível visualizar uma simulação no Rviz em [1].

[1] "RASTAR ON RVIZ", http://bit.ly/2reG4zN

Capítulo 4

AgrobPP

Têm-se na Figura 4.1 um esquema que pretende representar o funcionamento de todo o sistema, constituído por:

- o planeador global, responsável por gerar trajetórias seguras.
- o planeador local, que entra em ação assim que um obstáculo local é detetado, alterando a trajetória inicialmente gerada no planeador global.
- a divisão da trajetórias em curvas paramétricas de Bézier, utilizada para estimar a energia consumida.
- o planeamento "off-line"para postos de carregamento cujo propósito será detalhado no capítulo seguinte.

Nos seguintes pontos demonstram-se os processos que decorreram para desenvolver o AgrobPP, que consistiu em acrescentar a restrição do centro de massa do robô ao algoritmo A* restringido à orientação já existente.

4.1 Algoritmo A* restringido à orientação

Esta abordagem do algoritmo A* desenvolvido em [7] diferencia-se do A* tradicional pelo mapa e pela pesquisa dos vizinhos:

Mapa

Recebe um mapa de ocupação em duas dimensões, igual ao que um algoritmo A* tradicional utilizaria, mas transforma-o num mapa com três dimensões, sendo que a orientação representa a 3ª dimensão. Como demonstrado na Figura 4.2, o mapa de três dimensões é o resultado da multiplicação do mapa a duas dimensões por 16 camadas, sendo que cada camada representa uma gama de orientações. Com esta discretização, cada camada representa um ângulo de 22.5º.

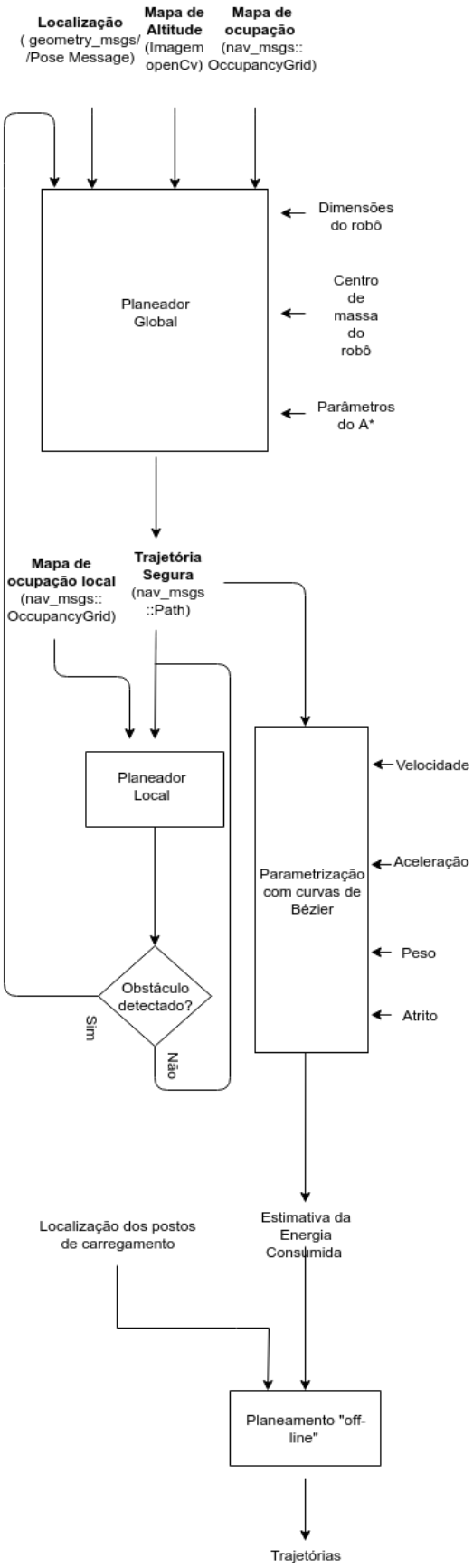

Figura 4.1: Esquema em blocos do AgrobPP

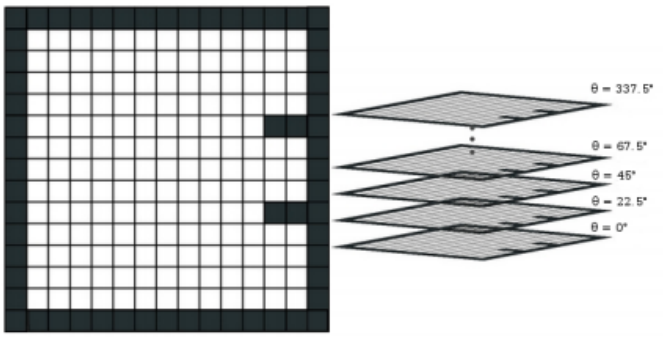

Figura 4.2: Mapa de entrada em 2D a sua representação em camadas [7]

Vizinhos

Tradicionalmente um A* tem conetividade 8, onde as 8 células vizinhas em volta da célula atual representam os vizinhos, ilustrado na Figura 4.3. Neste caso a conetividade é 16 como mostra a Figura 4.3, com o objetivo de conseguir representar todas as direções possíveis. Como se tem 16 camadas que representam 16 ângulos, é apropriado ter um vizinho para cada ângulo.

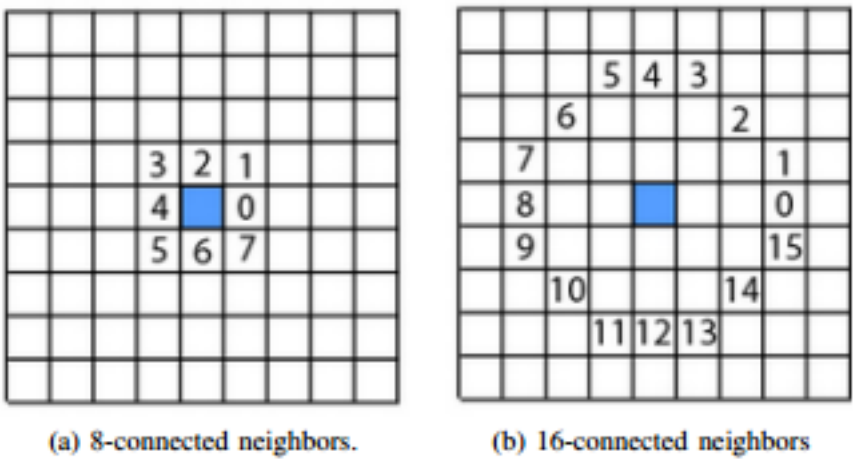

Figura 4.3: Representação dos vizinhos [7]

Apesar da conetividade de 16 células, apenas 6 são visitadas a cada iteração, para que sejam visitadas apenas células com orientações não muito diferente da orientação atual, θ, formando deste modo um caminho suave. Portanto, para um nó na camada n os seis vizinhos a visitar são:

- Dois vizinhos da camada que representam a orientação atual do robô (andar para a frente ou para trás)
- Dois vizinhos da camada $n-1$ que representam uma rotação de 22,5 º no sentido horário
- Dois vizinhos da camada $n+1$ que representam uma rotação de 22.5 º no sentido anti-horário.

Em cada camada, os nós visitados são aqueles que representam movimento naquela camada. Portanto, para a camada n, os nós visitados são os nós n e $n+8$, representando movimento com orientação θ e $\theta+180$.

Por exemplo, se a orientação do robô for $\theta = 0$, a camada atual é em $n = 0$ e:

- na camada 0, visitam-se os nós 0 e 8.
- na camada 15, visitam-se os nós 15 e 7
- na camada 1, visitam-se os nós 1 e 9

Isto implica que os possíveis movimentos do robô sejam com orientações de 0, 22.5 e 337.5. Tal caso está ilustrado na Figura 4.4 onde as os vizinhos visitados estão representados a verde. Nestes casos considera-se que o robô tem a capacidade de andar para frente e para trás, se tal não acontecer simplifica-se a procura para apenas 3 vizinhos, ou seja, na camada n apenas o nó n é visitado.

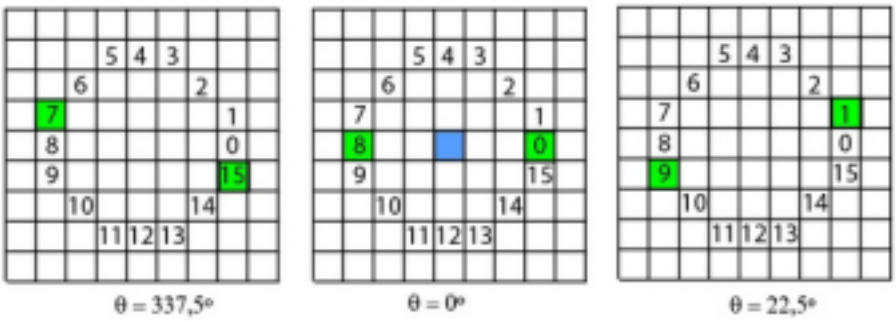

Figura 4.4: Vizinhos visitados para um nó na camada 0 [7]

Expansão de obstáculos

Como o próprio nome indica, a expansão dos obstáculos consiste em alargar o tamanho dos obstáculos existentes, normalmente através da pegada do robô, com dois objetivos: o robô pode ser expresso como um ponto evitando a necessidade de uma verificação completa de colisão, e elimina certas células da procura, otimizando a computação. Para perceber a importância deste processo, pode-se observar na Figura 3.6 da página 22 o resultado da aplicação de um A* sem qualquer expansão de obstáculos. Como é claramente visível, a colisão do robô com os obstáculos seria quase inevitável já que o caminho gerado percorre células tangentes a obstáculos.

Uma prática comum é expandir os obstáculos pelo raio do robô, o que funciona em robôs circulares ou quadrados, mas falha em retangulares. Se a distância do robô ao obstáculo for menor que o seu raio, então é certa uma colisão com o obstáculo. No caso de a distância ser maior, a colisão pode acontecer dependendo da sua orientação. A Figura 4.5 demonstra a importância da orientação na expansão dos obstáculos, em que é claro que para a mesma orientação um robô

de forma retangular tem diferentes distâncias do seu centro ao obstáculo, pelo que neste caso a expansão pelo raio do robô não iria funcionar. Portanto, a abordagem tomada em [7] utiliza a pegada atual do robô e funciona para qualquer tipo de robô. Em cada camada do mapa, a pegada do robô é calculada e com base nisso executa-se a expansão dos obstáculos. Cada camada do mapa terá a mesma informação sobre os obstáculos, mas diferentes expansões para os mesmos. Nas Figuras 4.6 e 4.7 mostra-se o resultado desta expansão aplicada a duas camadas do mapa da vinha de encosta, sendo que, os obstáculos estão representados a preto e a expansão representada a cinzento.

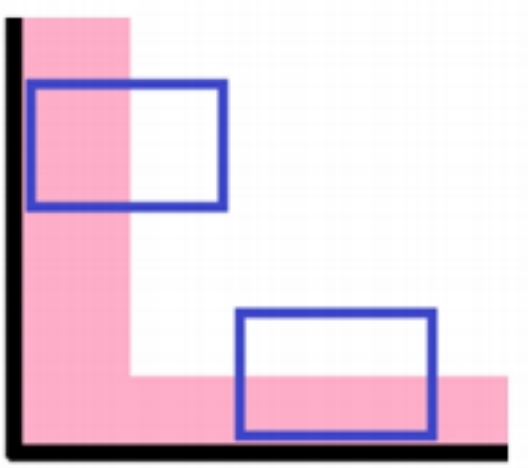

Figura 4.5: Expansão dos obstáculos dependendo da orientação [7]

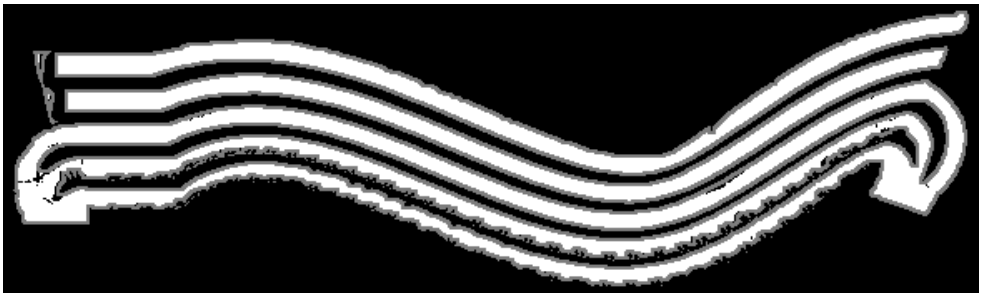

Figura 4.6: Expansão na camada 0 ($\theta = 0$)

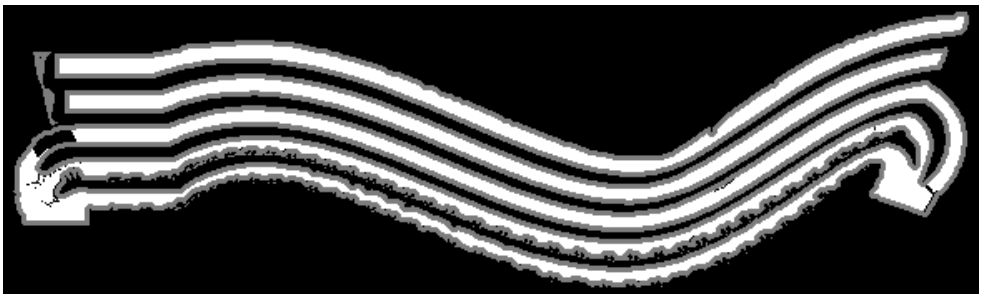

Figura 4.7: Expansão na camada 2 ($\theta = 50$)

4.2 Integração do Algoritmo A* com orientação

Este é o algoritmo que foi utilizado como base e modificado para que fosse possível incluir a inclinação do terreno. O código fornecido tinha duas componentes: o planeador com o algoritmo de pesquisa de caminho e o controlador com um controlador PD para que o robô pudesse seguir o caminho gerado. Daqui foi aproveitado o planeador, em particular a biblioteca de funções deste A* e o ficheiro com todas as estruturas criadas para armazenamento de dados, integrando o código no programa feito no capítulo 3 ficando assim com um nó em ROS que contém 2 algoritmos. O algoritmo A* tradicional, é tratado como RA* ("Relaxed A*"), e o A* com orientação ao qual se deu a denominação de CA*, uma vez que vem do projeto Carlos.

1. Neste primeiro teste foram considerados os seguintes parâmetros aleatórios:

   - **Dimensões:** 586x173 células
   - **Resolução:** 3 m/célula
   - **Dimensões do robô:** 0.2x0.1 m

   Tem-se na Figura 4.8 o resultado apenas com o laser frontal ativo, portanto, o robô apenas poderá andar em frente, e na Figura 4.9 o resultado com a marcha-atrás disponível.

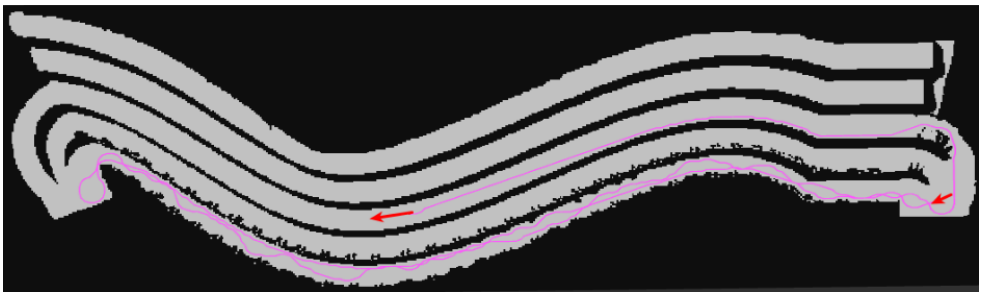

Figura 4.8: Teste 1 do CA*

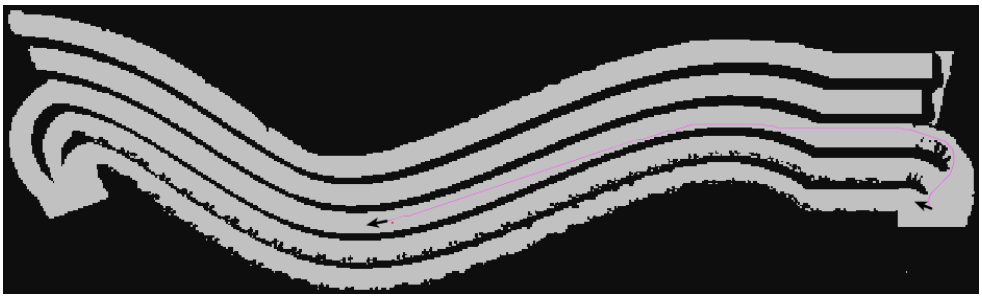

Figura 4.9: Teste 2 do CA*

No primeiro teste observa-se um caminho longo e complexo uma vez que, com a orientação inicial escolhida e sem marcha-atrás, o robô não teria espaço para efetuar a manobra naquele local, o que levou a que o tempo de execução fosse cerca de 128 segundos. Já no segundo teste o caminho gerado é mais curto, contudo, o tempo de execução do algoritmo foi ainda maior, 330 segundos, uma vez que com a marcha-atrás o CA* visita o dobro dos vizinhos.

2. No segundo teste utilizaram-se parâmetros mais próximos da realidade:

   - **Resolução:** 0.25 m/célula
   - **Dimensões do robô:** 1.2x0.8 m

A resolução foi escolhida com base nas características da plataforma Agrob14, que para fazer uma rotação de 90º descreve uma semicircunferência com 1 metro de raio. Ora, as características deste algoritmo implicam que sejam percorridas 4 células e 4 camadas para que ocorra uma rotação destas, portanto, o metro que o robô percorre nesta rotação é dividido pelas 4 células, obtendo uma resolução de 0.25 m/célula.

Na Figura 4.10 está ilustrado o resultado de um teste com condições mais favoráveis e sem marcha-atrás, que demorou cerca de 11 segundos a gerar o trajeto. Já na Figura 4.11 mostra-se o resultado de um teste com condições semelhantes ao primeiro teste e marcha-atrás disponível, tendo levado cerca de 340 segundos a gerar o caminho.

Concluí-se, portanto, que o algoritmo perde eficiência com a marcha-atrás disponível, devendo utilizar esta funcionalidade apenas em casos de extrema importância.

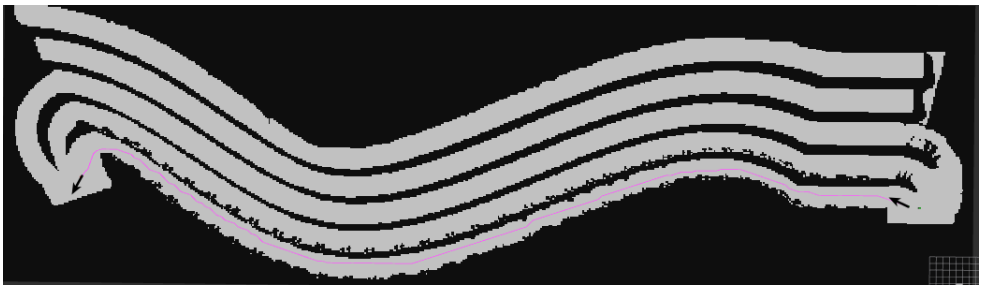

Figura 4.10: Teste 3 do CA*

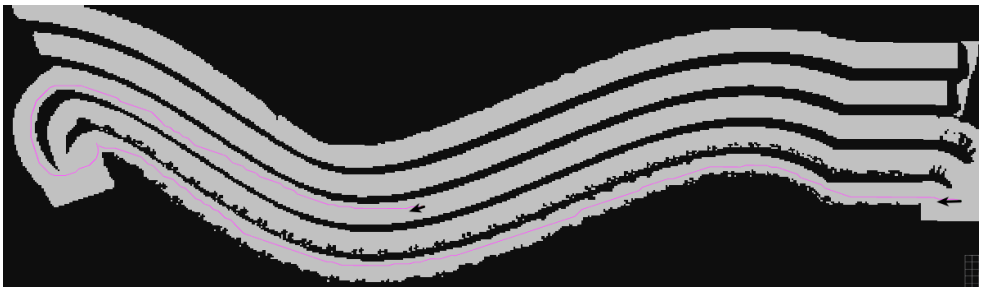

Figura 4.11: Teste 4 do CA*

4.3 Restrição ao centro de massa

A abordagem passou por determinar se numa certa posição do mapa a plataforma estaria ou não segura, não correndo o risco de cair e danificar o equipamento. Para tal, no inicio da execução do programa verifica-se a segurança de todas as células do mapa, sendo que as células que não são seguras passam a ser consideradas obstáculos. Para efetuar esta verificação,cada célula terá que conter informação sobre a sua altitude e inclinação.

Neste sentido, a primeira tarefa consistiu na criação de um mapa de altitude, a partir do qual se construiu um mapa de inclinação. Para estes mapas é utilizado o "openCV", uma vez que, como será explicado, o mapa de altitude armazenará dados numa gama de valores não suportada pela ferramenta "map server"do ROS.

4.3.1 Mapa de altitude

Este sistema passou então a ter como entrada um mapa de ocupação fornecido por um tópico em ROS, e um mapa de altitude fornecido por uma imagem e interpretado com recurso a "openCV". Tal mapa está ilustrado na Figura 4.12, onde é visível que a informação sobre a altitude está armazenada na cor de cada pixel e, cada pixel representa uma célula no mapa de ocupação em que preto corresponde à altitude mínima e branco à altitude máxima. Neste caso foi aplicado um fator de escala de 1/10, limitando a altitude máxima a 25,5 m. A leitura da altitude é feita durante a conversão do mapa de ocupação de duas dimensões no mapa de 16 camadas, guardando apenas a informação da altitude nas células livres de obstáculos. Neste mapa, observa-se que o primeiro socalco tem altitude 0, o segundo tem 12.5 metros e o terceiro tem 25 metros.

4.3.2 Mapa de inclinação

Com o mapa de altitude passamos a ter informação tridimensional sobre a localização de cada célula, o suficiente para definir um plano inclinado. Nesta etapa foi adicionada mais uma variável a cada célula, um vetor unitário normal à célula, tendo deste modo informação sobre a inclinação da mesma. O plano inclinado de cada célula é definido a partir de três pontos, sendo um deles a localização da própria célula e os outros dois as coordenadas de duas células vizinhas.

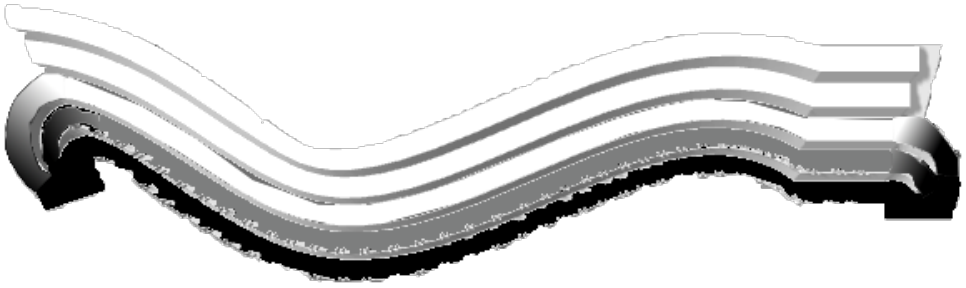

Figura 4.12: Mapa de Altitude

Pegando então num exemplo são definidos três pontos: $P1, P2, P3$. Em que $P1 = (x, y, z_1)$, é igual às coordenadas da célula atual, e $P2$ e $P3$ são as coordenadas de células vizinhas que são escolhidas de acordo com a sua disposição, disponibilidade e camada atual do mapa.

Para definir o plano na camada zero, considera-se a seguinte disposição de células:

$$\begin{array}{c} |\, P2 \,||\, P3 \,| \\ |\, P1 \,| \end{array}$$

onde $P2 = (x-1, y+1, z_2)$ e $P3 = (x+1, y+1, z_3)$.

Se por algum motivo não for possível fazer os cálculos com estes pontos, como por exemplo, as coordenadas do $P2$ estão fora dos limites do mapa ou representam uma célula com um obstáculo, são utilizadas outras disposições para definir o plano tais como:

$$\begin{array}{c} |\, P3 \,||P2 \,| \\ |P1 \,| \end{array} \qquad\qquad \begin{array}{c} |\, P2 \,||\, P3 \,| \\ |\, P1 \,| \end{array}$$

onde $P2 = (x-1, y+1, z_2)$ e $P3 = (x+1, y+1, z_3)$.

Na camada 12, por exemplo, que corresponde a uma orientação de -90º, é utilizada a seguinte disposição:

$$\begin{array}{c} |\, P2 \,| \\ |\, P1 \,| \\ |\, P3 \,| \end{array}$$

em que $P2 = (x+1, y+1, z_2)$ e $P3 = (x+1, y-1, z_3)$.

Daqui formam-se os vetores $\overrightarrow{v_1}$ e $\overrightarrow{v_2}$:

$$\overrightarrow{v_1} = P2 - P1 \tag{4.1}$$

$$\overrightarrow{v_2} = P3 - P1 \tag{4.2}$$

O produto vetorial entre estes dois vetores resulta num vetor normal à célula, $\overrightarrow{v_n} = (v_{nx}, v_{ny}, v_{nz})$:

$$\overrightarrow{v_n} = \overrightarrow{v_1} \times \overrightarrow{v_2} \tag{4.3}$$

Para poder visualizar esta operação, igualaram-se as coordenadas de cada vetor a uma variável RGB do "openCV"em que $B = v_{nx}, G = v_{ny}$ e $R = v_{nz}$. Está ilustrado na Figura 4.13 o mapa com informação sobre a inclinação, sendo que cada pixel tem uma cor de acordo com o vetor normal à célula correspondente. Como seria expectável todas as zonas planas têm cor vermelha, o que significa o vetor normal apenas tem componente em z e a célula não tem inclinação. As mudanças de cor nos troços de troca entre socalcos mostram que de facto essa zona tem inclinação.

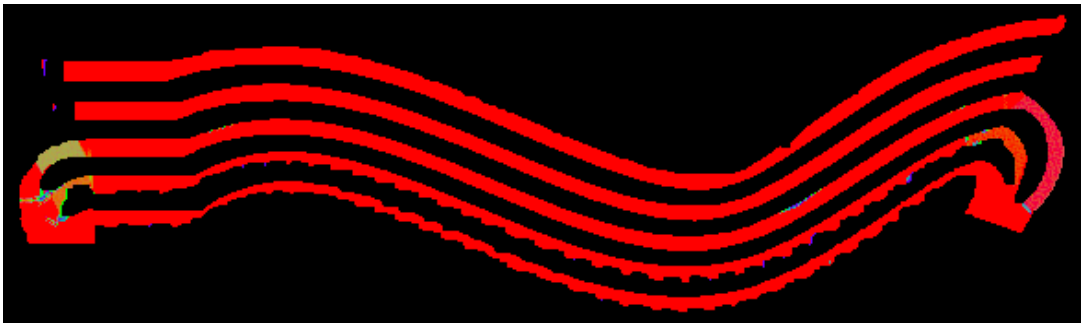

Figura 4.13: Mapa de Inclinação (Azul = v_{nx}, Verde = v_{ny}, Vermelho = v_{nz})

4.3.3 Verificação do centro de gravidade

Para determinar se determinada célula tem uma inclinação segura, começa-se por representar a sua inclinação através de ângulos de Euler, tendo então três componentes: "yaw"que representa uma rotação sobre o eixo z, "pitch"que representa a rotação em torno do eixo y e o "roll"que indica a rotação sobre o eixo x , como mostra a figura 4.14.

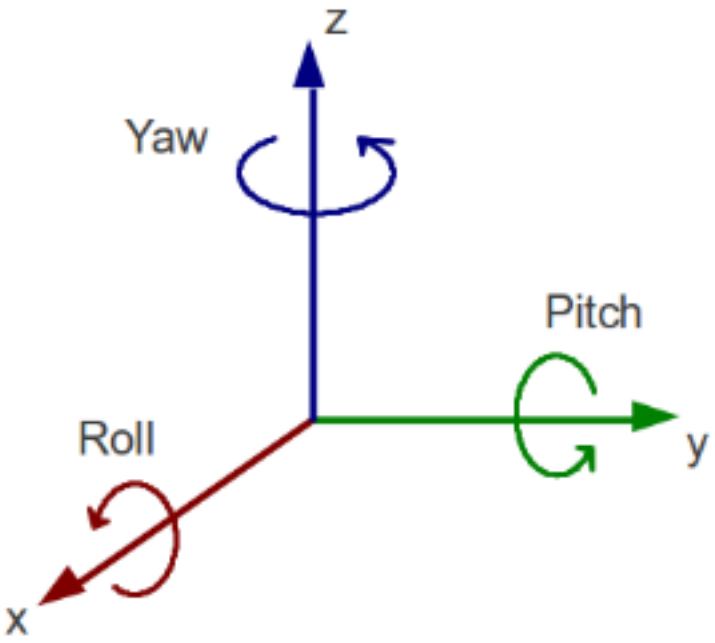

Figura 4.14: "yaw", "pitch"e "roll" [10]

Sendo o vetor normal $\overrightarrow{v_n} = (v_{nx}, v_{ny}, v_{nz})$:

$$pitch = \text{atan2}(v_{nx}, v_{nz}) \qquad roll = \text{atan2}(-v_{ny}, v_{nz}) \tag{4.4}$$

Já o "yaw"é dado pela orientação atual do robô, que para estes cálculos será a orientação correspondente à camada atual do mapa. Com estes dados constrói-se a seguinte matriz de rotação onde: $\alpha = yaw, \beta = pitch, \gamma = roll$

$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_y(\gamma) \tag{4.5}$$

$$R(\alpha,\beta,\gamma) = \begin{bmatrix} \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) & \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\ \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) & \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\ -\sin(\beta) & \cos(\beta)\sin(\gamma) & \cos(\beta)\cos(\gamma) \end{bmatrix}$$

Esta matriz serve para aplicar uma rotação ao centro de massa do robô para que este possa ser projetado no plano com a inclinação da célula atual. Aplica-se então uma rotação ao centro de massa representado pelo ponto $C_{massa} = (x_r, y_r, z_r)$ para obter a sua projeção no plano inclinado. As coordenadas são dadas por um referencial com origem no centro do robô.

$$Proj_{C_{massa}} = C_{massa}.R(\alpha,\beta,\gamma) \tag{4.6}$$

Se esta projeção do centro de massa ultrapassar as dimensões do robô, a célula é considerada perigosa e passa a ser lida como um obstáculo. Para tal, inicialmente apenas era feita a seguinte

verificação:

$$Se \mid Proj_{C_{massa}} \mid > \text{Dimensões, o robô irá cair}$$

Rapidamente se chega à conclusão que só funciona nos casos em que o centro de massa está localizado exatamente no centro da plataforma, ou seja, $Cmassa = (0,0,z_r)$. Por isso, foi então desenvolvido outro método para verificar a segurança do robô, constituído pelos seguintes passos:

1. Formar um retângulo com os vértices situados nas 4 rodas do robô

2. Aplicar uma rotação a cada um destes pontos com a matriz $R(\alpha,\beta,\gamma)$

3. Verificar se $Proj_{C_{massa}}$ está dentro do retângulo

    - Se estiver fora do retângulo, a célula atual passa a ser considerada um obstáculo

Um ponto $P = (x,y)$ está dentro de um retângulo definido pelos vértices A, B, e D se a seguinte expressão for verdadeira:

$$(0 < AP \cdot AB < AB \cdot AB) \wedge (0 < AP \cdot AD < AD \cdot AD)$$

Portanto, esta verificação é feita em todas as células de todas as camadas do mapa, utilizando as coordenadas x e y de $Proj_{C_{massa}}$, e as coordenadas de quaisquer 3 das 4 rodas do robô, considerando sempre que o referencial tem origem no centro da plataforma. Para demonstrar um exemplo foram considerados os seguintes valores:

Dimensões $= 120 \times 80\ cm$ (distância entre os eixos)

$Cmassa = (20,0,60)\ cm$

$\overrightarrow{v_n} = (0,41;0,41;1)$

$Pitch = 22 \quad Roll = -22$

A Figura 4.15 mostra o resultado da projeção deste ponto para as camadas 10 ($\theta = 225$) e 15 ($\theta = 337.5$) do mapa, onde é visível que na camada 10 o centro de gravidade está projetado fora dos limites do robô (representado pelos quatro pontos vermelhos), enquanto que na camada 15 o robô se encontra em segurança. As imagens com o resultado de todas as camadas estão disponíveis no anexo A.1.

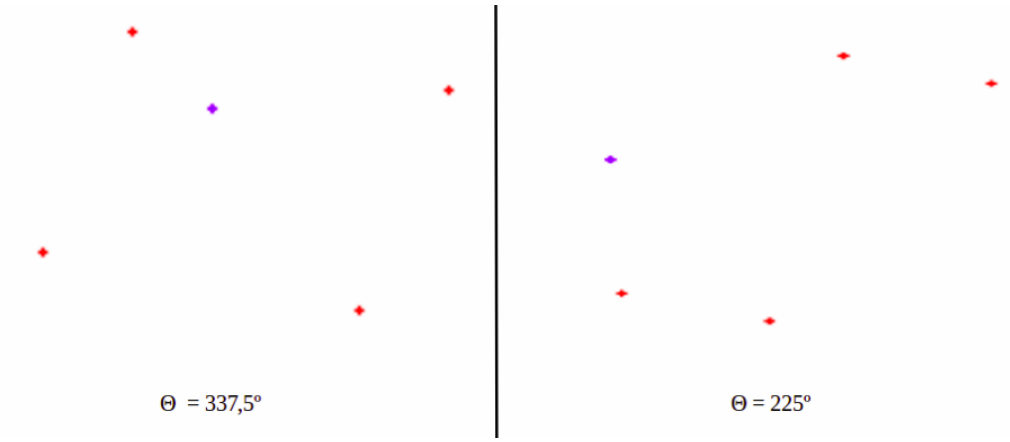

Figura 4.15: Centro de gravidade (a azul) num plano inclinado

Já numa zona sem inclinação onde $\overrightarrow{v_n} = (0,0,1)$, verifica-se pela Figura 4.16 que o centro de gravidade está sempre projetado na mesma zona. O resultado para as 16 camadas está também presente no anexo A.2.

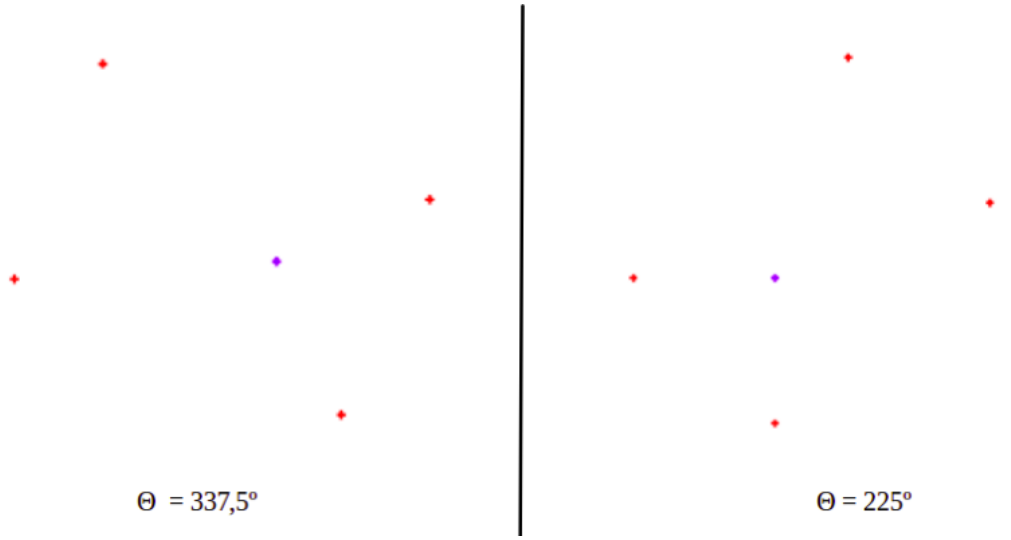

Figura 4.16: Centro de gravidade (a azul) num plano nivelado

Depois de aplicar esta operação a todas as células livres do mapa, fica-se com o mapa de 16 camadas completo e pronto para ser utilizado pelo algoritmo CA* . Para o mapa que tem vindo a ser utilizado com os parâmetros abaixo, todo este processo demora cerca de 20 segundos, tendo os resultados ilustrados na Figura 4.17 que mostra apenas uma zona com inclinação do mapa. As figuras com o resultado completo das 16 camadas do mapa estão presentes no anexo A.3.

- **Dimensões**: 586 x 173 (células)
- **Resolução**: 0.25 m/célula

- **Dimensões do robô**: 120 x 80 cm (distância entre os eixos)
- **Centro de massa**: (20,0,60) cm

Figura 4.17: Mapa de ocupação com inclinação

4.3.4 Resultados

Foram feitos alguns testes nas condições descritas anteriormente, ilustrados nas Figuras 4.18 e 4.19, e comparou-se o resultado com dois algoritmos já mencionados, o A* com orientação (CA*) e o "Relaxed A*"(RA*), tendo dados sobre tempo de execução e distância nas Tabelas 4.1 e 4.2

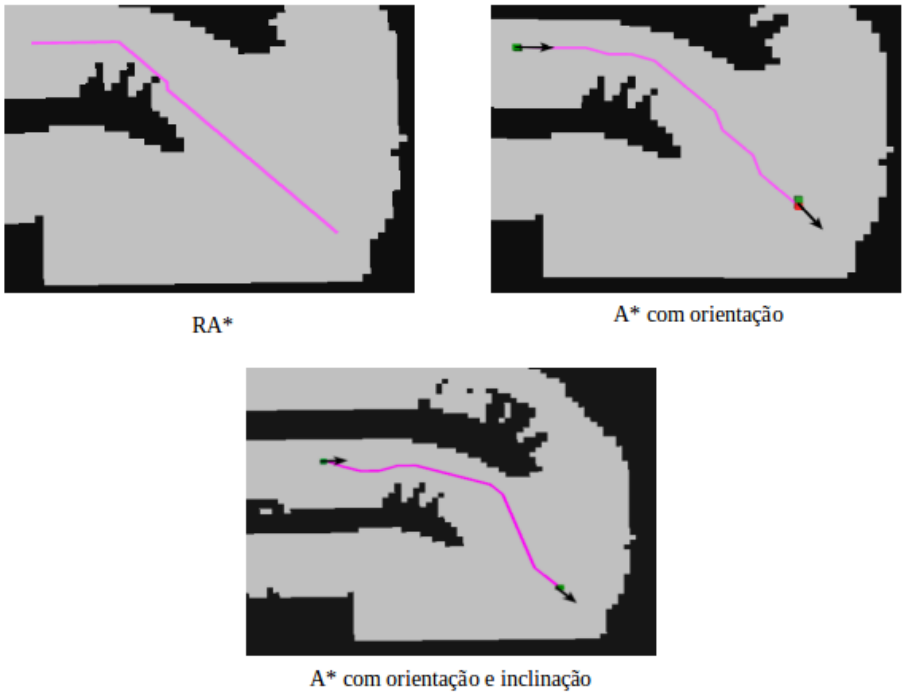

Figura 4.18: Teste 1 - Execução de diferentes algoritmos

Tabela 4.1: Teste 1 resultados

	RA*	CA*	CA* com inclinação
Tempo	5.8 μs	0.24 s	0.26 s
Distância	12.1 m	12.1 m	12.25 m

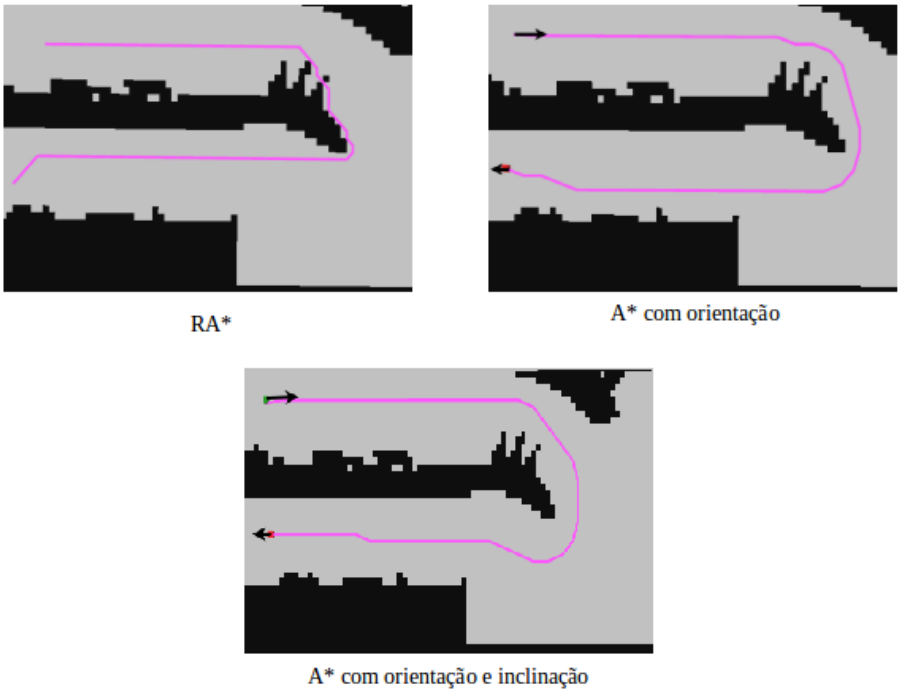

Figura 4.19: Teste 2 - Execução de diferentes algoritmos

Tabela 4.2: Teste 2 - Resultados

	RA*	CA*	CA* com inclinação
Tempo	5.27 ms	2.17 s	1.63 s
Distância	28 m	32.7 m	33 m

4.3.5 Desvio de obstáculos locais

Tudo o que foi feito até ao momento consistia num planeamento global da trajetória com os obstáculos estáticos definidos no início da execução do programa. Pretende-se agora que o algoritmo tenha ,também, a capacidade de modificar o caminho gerado quando surgir um obstáculo novo que interfira com a trajetória que o robô segue. Para tal o sistema terá mais um entrada, um mapa de ocupação local como por o exemplo o que está ilustrado na figura 4.20.

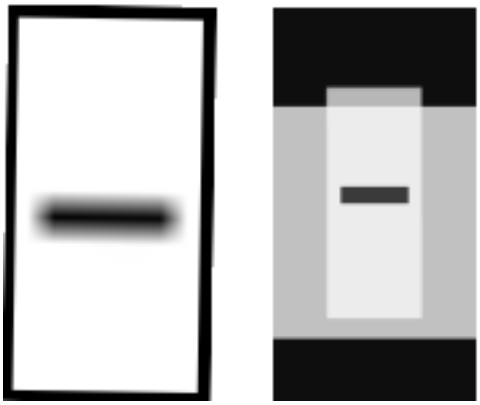

Figura 4.20: Exemplo de uma mapa local

Se o mapa local possuir um obstáculo, os procedimentos efetuados serão os seguintes:

- Verificar se o obstáculo local interfere com a trajetória do robô. Em caso positivo:

  1. Adicionar temporariamente o obstáculo ao mapa global
  2. Aplicar o processo de expansão ao obstáculo local
  3. Replanear a trajetória, sendo $\overrightarrow{P}$ o vetor que contém todos os pontos da trajetória, e $P[n]$ o ponto intersetado pelo obstáculo:
     (a) Considerar $P[n-i]$ como ponto de partida e $P[n+i]$ como ponto de destino, sendo que inicialmente $i = 1$
     (b) Executar o A*
     (c) Se o caminho não for encontrado, incrementar i ($i = i+1$) e voltar a 3a (o ponto de partida estará limitado à posição do atual do robô)
     (d) Se este método falhar, é porque não é possível contornar o obstáculo em segurança
  4. Se 3 foi executado com sucesso:
     (a) Substituir a gama de pontos desde $P[n-i]$ a $P[n+i]$ do trajeto atual, pelo caminho gerado no planeamento local
     (b) Apagar o obstáculo local do mapa global

Na Figura 4.21 tem-se alguns resultados desta operação, podendo visualizar o caminho original e o caminho planeado localmente, onde Δt é o tempo de execução das quatro tarefas mencionadas.

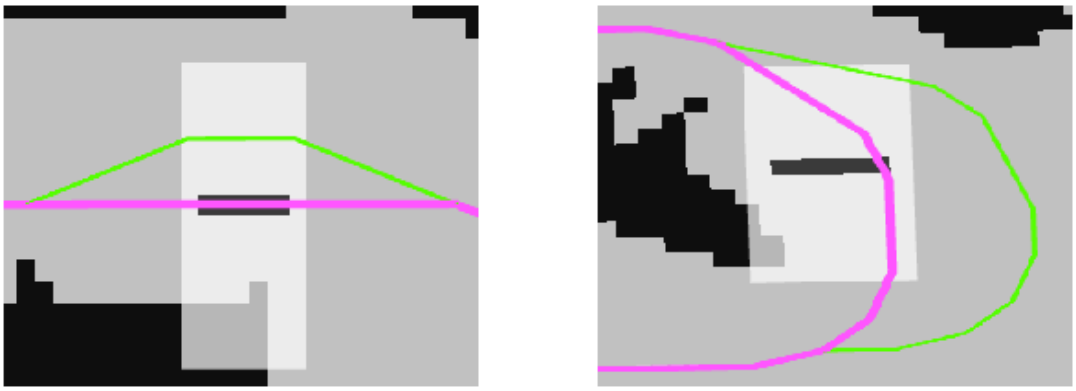

Figura 4.21: Planeamento local de trajetória

Capítulo 5

Planeamento "Off-line"para Postos de Carregamento

Uma vez que as baterias de robô têm autonomia limitada, será útil possuir postos de carregamento localizados em diferentes zonas da vinha, e ter o robô apto para se deslocar até um destes postos de forma autónoma sempre que detetar que os níveis das baterias são inferiores a um nível predefinido. Neste sentido, o presente capítulo apresenta um método para escolher o posto mais adequado ao carregamento, não tendo, como já foi dito em outros capítulos, necessariamente que ser o posto mais próximo, mas sim o posto cujo caminho exija o menor custo energético à plataforma robótica.

5.1 Planeamento

Em situações em que se recebe um alerta de bateria fraca, pretende-se que o robô tenha acesso a uma trajetória de forma quase instantânea, sem ter que perder tempo a processar o seu planeamento. Portanto, o planeamento será previamente feito em modo "off-line", guardando todos as trajetórias geradas para que robô tenha acesso quase instantâneo às mesmas.

O planeamento será feito com recurso ao algoritmo desenvolvido anteriormente, sem qualquer alteração ao mesmo. O conceito apresentado em [18] consiste em fazer o planeamento "off-line"de todos os caminhos possíveis e armazenar em cada célula do mapa um caminho que tem como origem a própria célula e como destino o posto de carregamento mais adequado, ignorando a orientação do ponto de destino, uma vez que nas proximidades do posto o robô passa a navegar com recurso a "tags"visuais. Como o algoritmo utiliza 16 camadas do mapa, cada célula terá que armazenar 16 caminhos, um para cada orientação do robô. Este processo é demorado, mas o tempo de processamento não é muito relevante uma vez que o processo é feito em modo "off-line"e apenas precisa de ser executado uma vez para cada mapa e diferente tipo de robô.

Cada posto de carregamento está localizado no centro, (a,b) de um circulo paramétrico, de raio r variável (equação 5.1), desde 0 a um máximo de 2 metros, uma vez que nesta zona o robô

terá a capacidade de navegar sem recurso ao A*.

$$\begin{cases} x &= a + r\cos(t),\ 0 \leq t \leq 2\pi \\ y &= b + r\sin(t),\ 0 \leq t \leq 2\pi \end{cases} \tag{5.1}$$

Neste sentido, serão realizados os seguintes passos:

1. Em cada célula e em cada camada:

   (a) Planear um caminho desde a célula atual com orientação da camada atual, até um ponto (x,y) da circunferência , considerando $r = 0$ e ignorando a orientação final . Caso o A* não encontre um caminho, aumenta-se o r e executa-se novamente A* para cada um dos pontos da circunferência até que um caminho seja encontrado.

   (b) Estimar o custo energético de cada um dos caminhos e selecionar apenas o que tem menor custo, guardando esse caminho na célula atual

   (c) Percorrer todas as células ocupadas pelo caminho escolhido, guardando esse mesmo caminho em cada célula para não ter necessidade de fazer um planeamento em todas as células do mapa

5.2 Custo Energético

Em [16] é apresentado um método para calcular o custo energético que um caminho irá gerar. Para tal, a trajetória é descrita por curvas de Bézier cúbicas, tal como já foi indicado na página 15. Seguindo este método, determina-se uma curva de Bézier entre cada dois pontos seguidos da trajetória, e utiliza-se a informação dessa curva para derivar aceleração e velocidade, tendo então dados para fazer o cálculo apresentado na equação 5.2 proposta em [16].

$$E_{i-i,i} = ksT_{i-1,i} + m \sum_{j=1}^{T_{i-1,i}/\Delta_t} (\mu g + a_j) v_j \Delta_t \tag{5.2}$$

Onde:

- $T_{i-1,i}$ é o tempo de viagem entre os ponto w_{i-1} e w_i, sendo $w_i = (x_i, y_i)$
- ks é um coeficiente estático que representa a energia perdida na transformação de energia elétrica em energia mecânica
- $m =$ massa do robô
- μ é o coeficiente de atrito, e g a aceleração da gravidade
- a é aceleração do robô e v a sua velocidade
- Δ_t é uma pequena variação de tempo escolhida para a discretização

As curvas cúbicas são normalmente utilizadas para suavizar a trajetória, mas como este algoritmo já está construído para formar caminhos suaves, optou-se por utilizar curvas lineares de Bézier (equação 5.3) sempre que uma série de pontos tem a mesma orientação. Quando a orientação muda de ponto para ponto, utilizam-se curvas quadráticas de Bézier (equação 5.4).

$$\begin{cases} x(u) &= (1-u)x_{i-1} + ux_i \\ y(u) &= (1-u)y_{i-1} + uy_i \end{cases} \tag{5.3}$$

$$\begin{cases} x(u) &= ((1-u)^2)x_{i-1} + 2u(1-u)x_a + u^2x_i \\ y(u) &= ((1-u)^2)y_{i-1} + 2u(1-u)y_a + u^2y_i \end{cases} \tag{5.4}$$

O ponto (x_a, y_a) é determinado pela intersecção das retas que passam nos pontos w_i e w_{i-1} definidas por um ponto e um ângulo. O ângulo corresponde à orientação do robô no ponto da trajetória, algo que é fornecido pelo algoritmo A* do AgrobPP.

O parâmetro u varia entre 0 e 1, e é determinado de modo a que a velocidade linear ao longo do percurso se mantenha constante, estando este processo demonstrado nas seguintes equações:

$$v = \sqrt{\dot{x}(u)^2 + \dot{y}(u)^2} \tag{5.5}$$

Assume-se que a velocidade v tem o valor constante de k:

$$k^2 = \dot{x}(u)^2 + \dot{y}(u)^2 \tag{5.6}$$

$$= \left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2 \tag{5.7}$$

$$= \left(\frac{\mathrm{d}x}{\mathrm{d}u}\right)^2 \left(\frac{\mathrm{d}u}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}u}\right)^2 \left(\frac{\mathrm{d}u}{\mathrm{d}t}\right)^2 \tag{5.8}$$

$$= \left[\left(\frac{\mathrm{d}x}{\mathrm{d}u}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}u}\right)^2\right] \left(\frac{\mathrm{d}u}{\mathrm{d}t}\right)^2 \tag{5.9}$$

$$\Leftrightarrow \frac{\mathrm{d}u}{\mathrm{d}t} = \pm \sqrt{\frac{k^2}{\left(\frac{\mathrm{d}x}{\mathrm{d}u}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}u}\right)^2}} \tag{5.10}$$

No domínio discreto, para j = 0,1,2,3,4..., j_f :

$$u_{j+1} = u_j + \sqrt{\frac{k^2}{\left(\frac{\mathrm{d}x}{\mathrm{d}u}\right)^2 |_{u=u_j} + \left(\frac{\mathrm{d}y}{\mathrm{d}u}\right)^2 |_{u=u_j}}} \Delta t \tag{5.11}$$

A energia gasta entre os pontos w_i e w_{i-1} é então calculada com uma pequena adaptação da equação 5.2 de modo a que se tenha em conta a inclinação do terreno:

$$E_{i-i,i} = m\sum_{j=1}^{j_f}(\mu g_{z_j} + a_j)v_j\Delta_t - m\sum_{j=1}^{j_f}(\mu g_{x_j} + a_j)v_j\Delta_t \tag{5.12}$$

Como num plano inclinado a gravidade tem 3 componentes, $\overrightarrow{g} = (g_x, g_y, g_z)$, dividiu-se a fórmula de acordo com duas destas componentes, em que a gz contabiliza a força gravítica perpendicular ao robô , e gx a força que está na direção do movimento do robô. Esta componente é de sinal variável para incluir subidas ou descidas.

A velocidade e aceleração são calculadas a partir da primeira e segunda derivada das equações de Bézier, tendo as seguintes expressões no domínio discreto:

$$v_j = \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2}/\Delta_t \tag{5.13}$$

$$a_j = \sqrt{(v_{j+1} - v_j)^2 + (v_{j+1} - v_j)^2}/\Delta_t \tag{5.14}$$

Para o cálculo da energia falta apenas informação sobre o vetor $\overrightarrow{g}$. Ora, sabe-se que em qualquer zona plana $\overrightarrow{g} = (0,0,g_z)$ portanto, para determinar o vetor em zonas inclinadas executam-se os passos descritos abaixo, que correspondem ao inverso do processo de verificação do centro de gravidade, exposto na página 32.

1. Determinar o *pitch*, *roll* e *yaw* associado à célula em questão
2. Utilizar uma matriz inversa à matriz 4.5 da página 33 para rodar o vetor $(0,0,g_z)$ para o plano do robô:

$$\overrightarrow{g} = (0,0,g_z) \cdot R(\alpha,\beta,\gamma)^{-1} \tag{5.15}$$

Por exemplo, numa célula em que: $yaw = 90$ $pitch = 26.57$ $roll = -26.57$ $g_z = 9.8$

$$\overrightarrow{g} = (0,0,9.8) \cdot R(90, 26.57, -26.57)^{-1}$$
$$\overrightarrow{g} = (-4.38; 3.92; 7.84)$$

a componente g_x é negativa porque este exemplo foi retirado de uma trajetória durante uma subida. Foram efetuados alguns testes em "Matlab", expostos no sub-capítulo seguinte, para validar esta operação.

5.2.1 Testes em Matlab

Recorreu-se ao "Matlab"para validar a transformação da trajetória formada por vários pontos em curvas de Bézier. Para tal, é fornecido um ficheiro do tipo "csv"com os seguintes dados sobre

um determinado trajeto: coordenada x, y, orientação ($yaw, pitch\ e\ roll$), e aceleração gravítica. Estes dados podem facilmente ser visualizados no "Matlab", tendo na Figura 5.1 o caminho gerado pelo algoritmo e na Figura 5.2 os pontos da trajetória bem como as curvas de Bézier resultantes representados no "Matlab". O código "Matlab"produzido nesta tarefa está disponível no anexo B.1

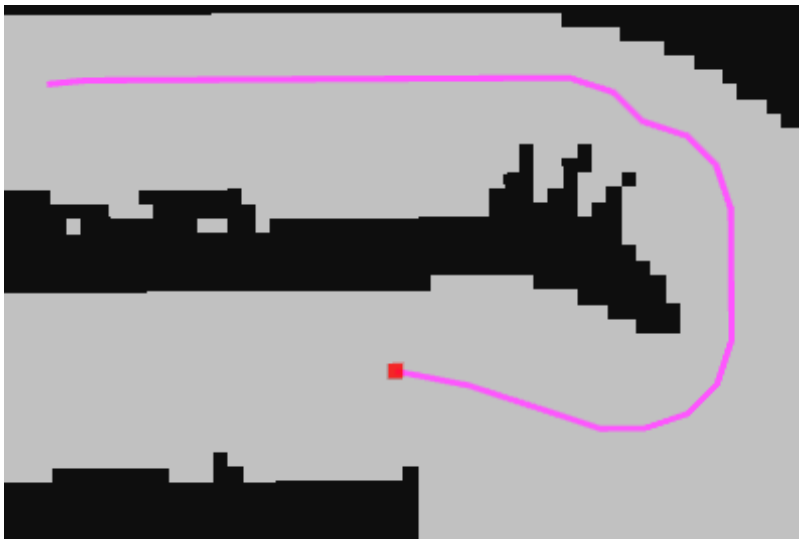

Figura 5.1: Trajeto a ser representado em Matlab

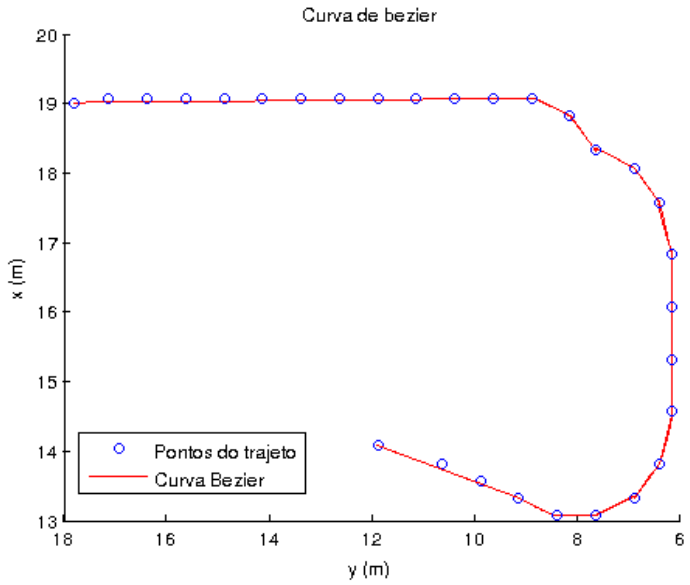

Figura 5.2: Curvas de Bézier em "Matlab"

Posto isto é também possível confirmar com recurso ao gráfico presente na Figura 5.3 que tal como pretendido, a velocidade permanece constante em todo o trajeto.

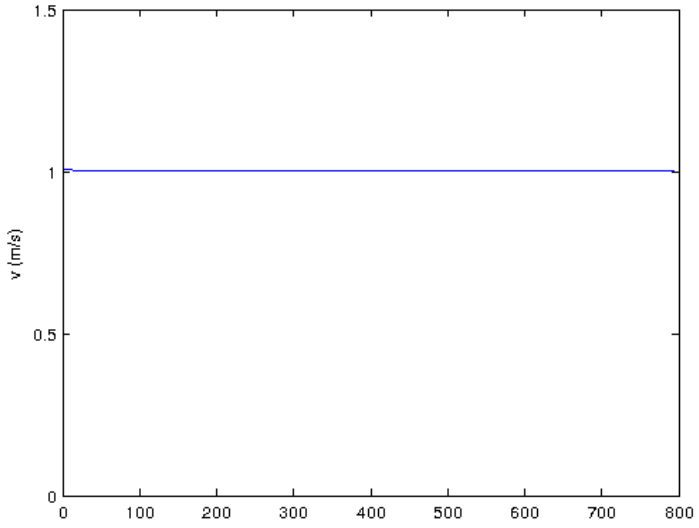

Figura 5.3: Velocidade ao longo do trajeto

Para verificar a rotação do vetor $\overrightarrow{g}$, fez-se uma representação gráfica das componentes (g_x, g_y, g_z) da aceleração gravítica, juntamente com a altitude e orientação do robô ao longo do percurso. Tendo na Figura 5.4 um exemplo de uma descida, e na Figura 5.5 uma subida. É visível que ao longo da descida a componente g_x toma valores positivos, enquanto que ao longo de uma subida toma valores negativos.

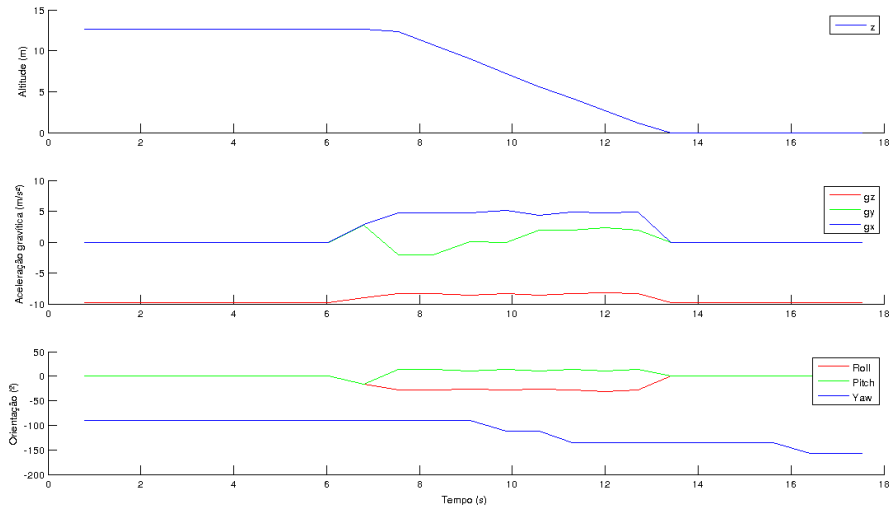

Figura 5.4: Verificação da rotação de 'g' numa descida

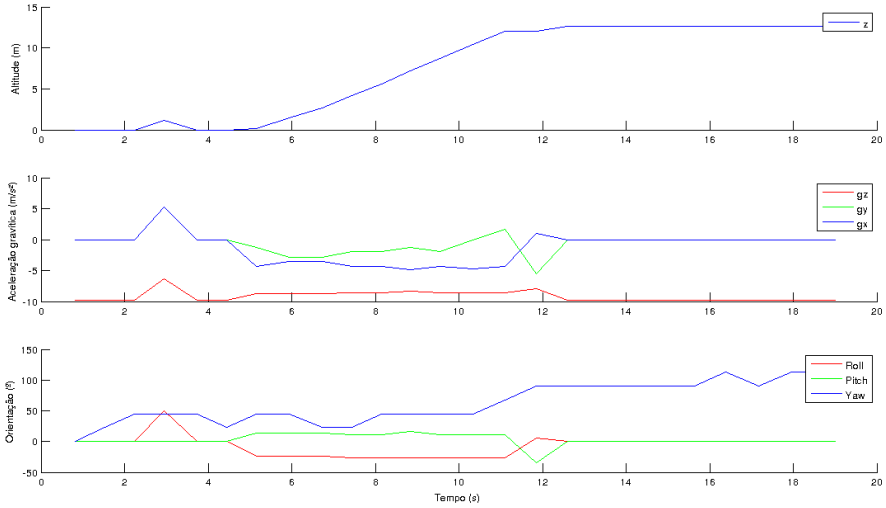

Figura 5.5: Verificação da rotação de 'g' numa subida

5.2.2 Estimação do consumo energético

As figuras abaixo mostram vários caminhos gerados com sentidos opostos (subida ou descida) com os resultados da estimação do consumo energético disposto em tabelas, sendo que para cada caso se efetuaram três testes, contabilizando a média final.

Para tal consideraram-se as variáveis que se seguem, estando o centro de massa no centro do robô apenas para que o algoritmo não altere significativamente o caminho gerado em caso de troca de sentido:

$$m = 1\ kg\ \mu = 0.06\ g = 9.8\ ms^{-2}\ v = 1\ ms^{-1}$$
$$a = 0\ ms^{-2}\ \Delta t = 0.001s\ C_{massa} = (0,0,0.4)m$$

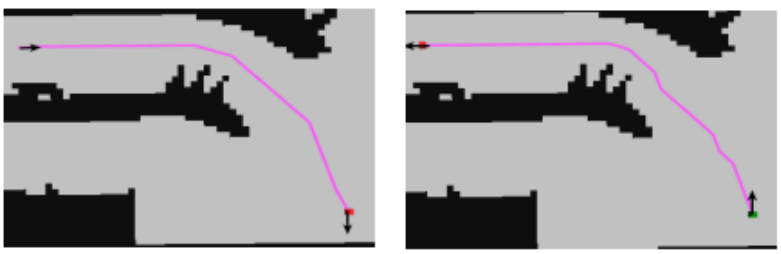

Figura 5.6: Descida (à esquerda) e subida (à direita) de 19 metros

Figura 5.7: Descida (à esquerda) e subida (à direita) de 9 metros

Tabela 5.1: Cálculo energético

Energia (J)	19 m		9 m	
	Descida	Subida	Descida	Subida
1	8.5	12	3.7	4.7
2	8.4	11	3.6	4.9
3	8.48	10.95	3.3	4.8
Média	8.46	11.32	3.53	4.8

Observa-se pela tabela 5.1 que, tal como o esperado, as descidas exigem um consumo energético menor que as subidas. Têm-se ainda na Figura 5.8, um gráfico semelhante ao apresentado na Figura 5.5 com a representação da altitude, aceleração gravítica, e energia consumida para o caso da descida de 19 metros, onde se pode visualizar a evolução da energia consumida ao longo do trajeto. Já a Figura 5.9 ilustra os mesmos dados para um caminho no sentido oposto, ou seja, uma subida. Comparando os dois gráficos, verifica-se que na subida a energia consumida sobe de forma aproximadamente linear, enquanto que na descida se observam algumas zonas de atenuação.

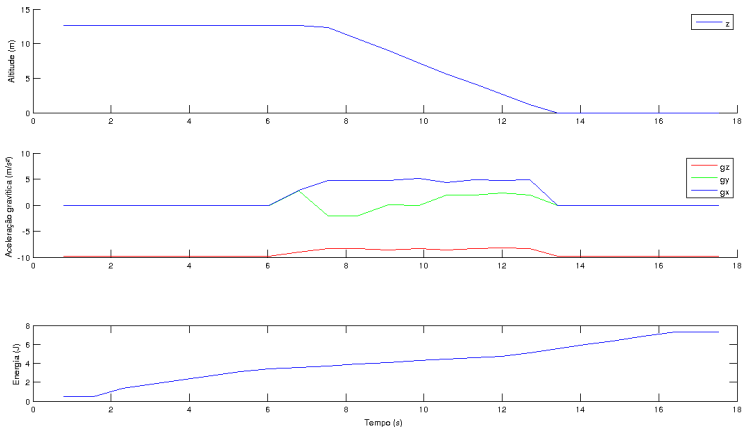

Figura 5.8: Gráfico de uma descida de 19 m

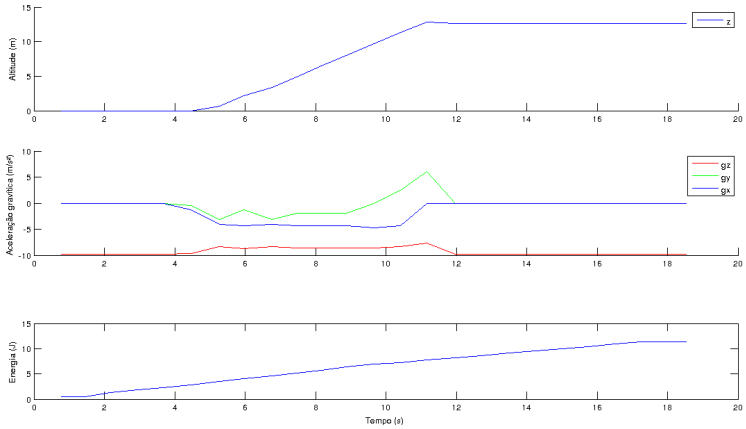

Figura 5.9: Gráfico de uma subida de 19 m

5.2.3 Método alternativo para estimar um custo energético

Este método consiste em considerar que o custo apenas é composto pela distância do caminho e a diferença entre as cotas (altitude). É um método simples e de fácil implementação que serve para o que é pretendido neste contexto uma vez que, deste modo o caminho escolhido não será obrigatoriamente o de menor distância, mas sim aquele que for constituído por mais descidas, que por si só pressupõem um custo energético menor.

Com isto pode-se optar por alterar a função custo do A* para que esta inclua a diferença de cotas passando o custo a ter a expressão descrita em 5.16, em que n é o ponto atual, n_0 o ponto de origem e n_f o destino:

$$G(n) = custo_{n_0,n} - (cota_{n_0} - cota_n) \cdot Ks,\ se\ (cota_{n_0} - cota_n) \leq 0 \tag{5.16}$$

$$H(n) = custo_{n,n_f} - (cota_n - cota_{n_f}) \cdot Ks,\ se\ (cota_{n_0} - cota_n) \leq 0 \tag{5.17}$$

$$\tag{5.18}$$

$$G(n) = custo_{n_0,n} - (cota_{n_0} - cota_n) \cdot Kd,\ se\ (cota_{n_0} - cota_n) > 0 \tag{5.19}$$

$$H(n) = custo_{n,n_f} - (cota_n - cota_{n_f}) \cdot Kd,\ se\ (cota_{n_0} - cota_n) > 0 \tag{5.20}$$

Como se observa na equação 5.16, o custo varia consoante o sinal da diferença de altitude, ou seja, subir ou descer não terá o mesmo custo. Para evitar situações como a que está exposta na Figura 5.11, Kd tem que ser positivo e inferior a 1, caso contrário uma grande descida terá um custo menor que uma pequena distância em terreno nivelado. Ks apenas deverá ser maior que Kd , deste modo subir terá um custo superior a uma descida. O valor a escolher para Kd depende do peso que se pretende que uma subida, que não deve ser muito elevado, caso contrário a mínima subida provoca uma grande alteração no algoritmo gerado. Se Kd e Ks fossem iguais, o custo de uma subida iria ser anulado por uma descida, mantendo apenas o critério da distância mínima.

Têm-se na figura 5.10 um exemplo de um caminho a subir a subir e um caminho a descer gerado com este critério e com o critério da distância mínima, onde é visível que no caso da subida são escolhidos pontos onde a subida é menos acentuada. Na descida, o caminho não se distância muito do original uma vez que o seu peso é menor. Para tal, considerou-se que $Kd = 0.6$ e $Ks = 1$.

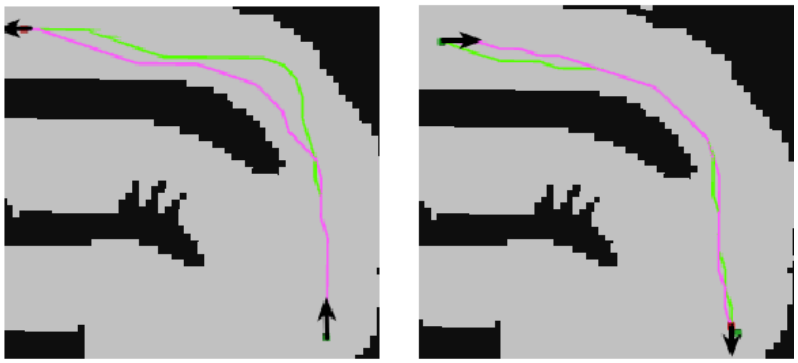

Figura 5.10: Verde - Caminho gerado com cotas ; Roxo - caminho gerado pela distância mínima

Com este método o custo entre o ponto w_{i-1} e w_i é dado por:

$$C_{i-1,i} = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} - (cota_{w_{i-1}} - cota_{w_i}) \cdot Ks,\ se\ (cota_{w_{i-1}} - cota_{w_i}) \leq 0 \tag{5.21}$$

$$C_{i-1,i} = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} - (cota_{w_{i-1}} - cota_{w_i}) \cdot Kd,\ se\ (cota_{w_{i-1}} - cota_{w_i}) > 0 \tag{5.22}$$

Esta abordagem é útil caso seja necessário escolher entre um caminho que sobe ou um caminho que desce sendo que para tal, as constantes Kd *e* Ks devem seguir a regras mencionadas anteriormente. Caso contrário corre-se o risco da ocorrência de situações como a que está ilustrada na Figura 5.11, em que faria todo o sentido que o caminho verde fosse o escolhido, no entanto, se $Kd \geq 1$ o custo do caminho roxo será menor que o custo do caminho verde, levando o algoritmo a indicar um caminho que apesar de possuir uma descida, implicará um maior custo energético ao robô.

Figura 5.11: Exemplo de dois caminhos gerados até postos de carregamento

A Tabela 5.2 mostra os resultados para os caminhos testados anteriormente com a estimação de energia consumida (Figuras 5.6 e 5.7), onde é visível que as descidas têm sempre um custo menor.

Tabela 5.2: Custos em subidas ou descidas com diferença de cotas

Custo	19 m		9 m	
	Descida	Subida	Descida	Subida
1	9.4	29.6	3.0	15.8
2	9.3	29.8	2.7	15.7
3	9.4	28.8	2.7	15.5
Média	9.36	29.4	2.8	15.67

Estes valores não podem ser interpretados como a energia consumida pelo robô. São apenas uma estimativa do custo que a plataforma terá para atingir um determinado ponto tendo em conta a distância e as diferenças de altitude. É um método muito menos complexo que a estimação de energia com curvas de Bézier, e que pode ser utilizado na escolha do melhor caminho até um posto de carregamento, contudo, o resultado final não é tão preciso.

5.3 Resultados

Para demonstrar o resultado, foi atribuída uma cor a cada célula conforme o posto de carregamento escolhido, em que azul representa o posto 1, verde o posto 2 e vermelho o posto 3. Cada posto está localizado num socalco da vinha para poder ter pelo menos duas zonas com inclinação, em que o posto 1 está a 0 metros de altitude, o posto 2 a 12,5 metros e o posto 3 a 25 metros. A localização destes postos está exposta na Figura 5.12, em que cada posto se representa pelo seu número (1, 2 e 3). Esta figura representa apenas uma porção do mapa de ocupação da vinha utilizado para testar o planeamento "off-line"com dimensões de 100x55 células, cerca de 5% do mapa completo.

Efetuaram-se testes para as camadas 0, 4, 8 e 12 do mapa, correspondentes a orientações de 0º, 90º, 180º e 270º.

Figura 5.12: Localização dos postos de carregamento

5.3.1 Planeamento "off-line"com estimação da energia consumida

Os resultados desta abordagem estão ilustrados na figura 5.13, tendo levado cerca de 80 minutos a executar a tarefa para as quatro camadas representadas. As células a preto representam obstáculos ou zonas onde não foi possível gerar um caminho para qualquer um dos três postos. É também importante notar que se for escolhido um caminho mais longo, não será obrigatoriamente por este ser o de menor custo energético, mas por ser o único que o A* encontrou.

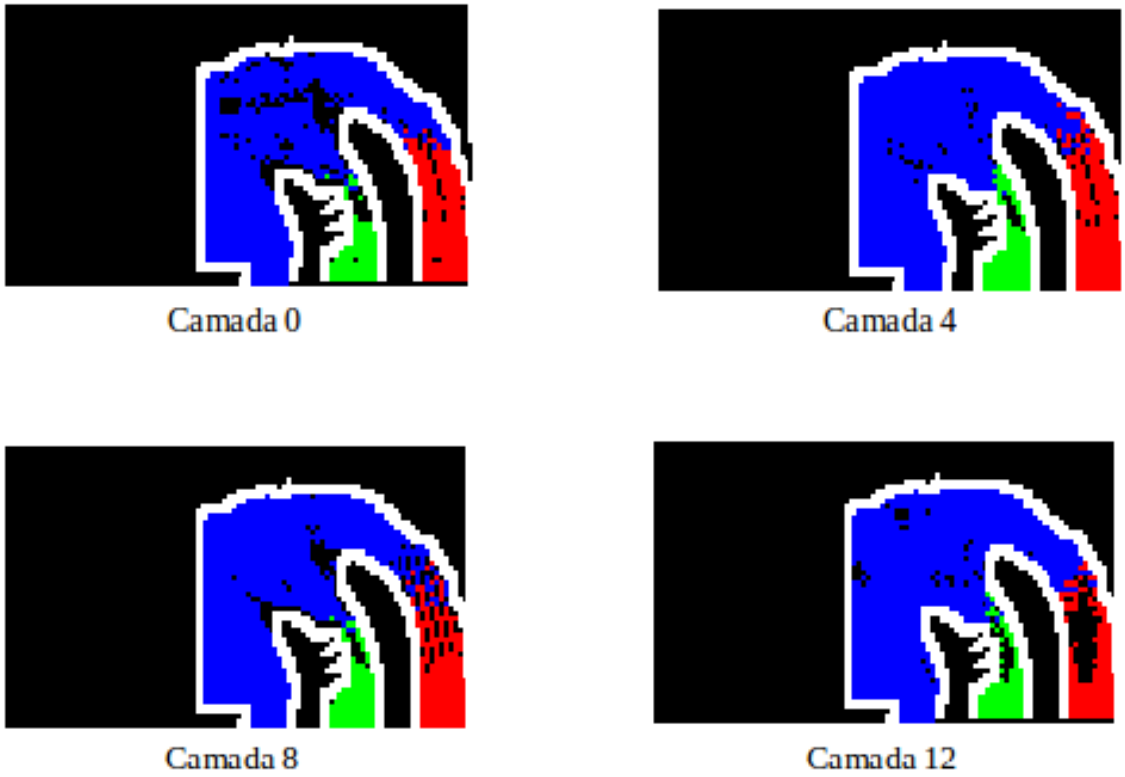

Figura 5.13: Resultados do planeamento "off-line"com estimação de energia

5.3.2 Planeamento "off-line"com método alternativo

Os resultados desta abordagem estão ilustrados na figura 5.14, tendo levado cerca de 80 minutos a executar a tarefa para as quatro camadas representadas. Comparando as duas abordagens, observa-se que os resultados são semelhantes com a diferença que este método tem preferência por caminhos com descidas já que assume que uma descida implica um custo energético menor, algo que nem sempre é verdade, uma vez que os equipamentos não estão preparados para regenerar energia. Portanto, se uma descida implicar uma distância percorrida muito maior que uma subida ou uma zona plana, o seu custo energético será superior.

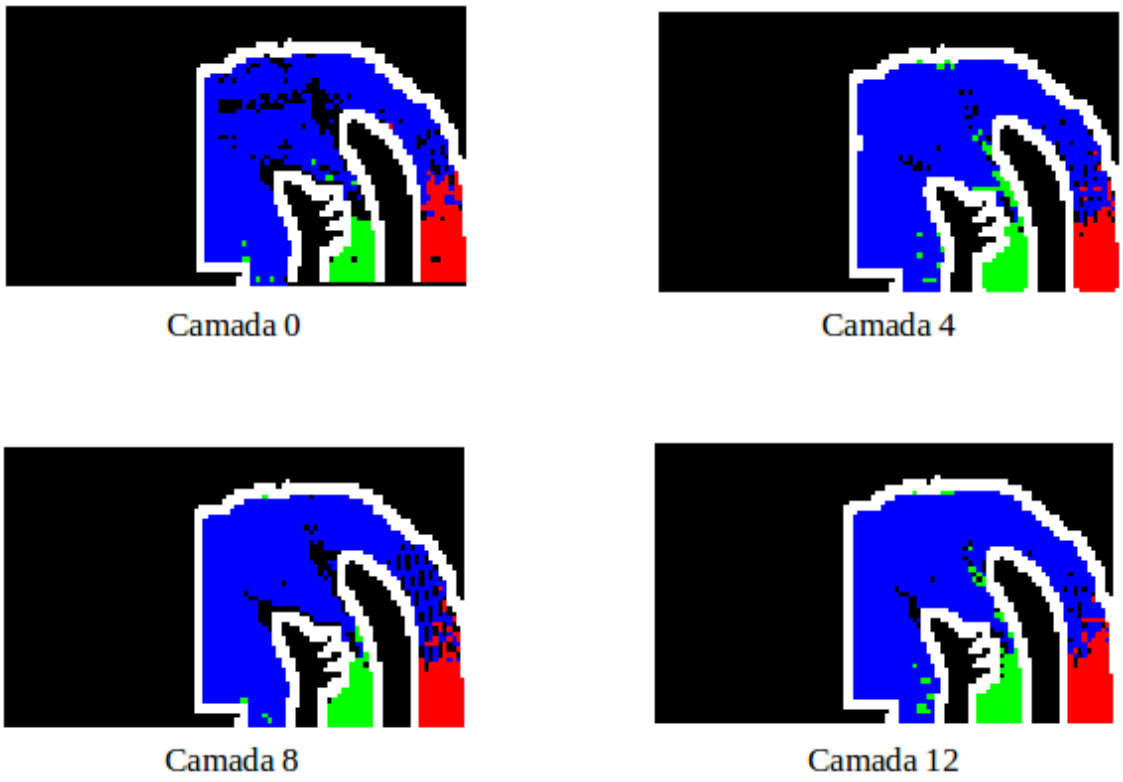

Figura 5.14: Resultados do planeamento "off-line"com método alternativo

Capítulo 6

Conclusões e Trabalho Futuro

O objetivo desta dissertação consistia em encontrar uma solução de planeamento de trajetórias seguras para as vinhas de encosta, de modo a que a transição do robô entre os socalcos se efetuasse de forma autónoma e segura. Para tal, começou-se por analisar vários métodos para planeamento de trajetórias dando-se ênfase a métodos que recorrem a mapas de grelha, em particular o A*.

Decidiu-se então utilizar um algoritmo A* restringido à orientação que fora desenvolvida para outro projeto com ferramentas semelhantes às utilizadas nesta dissertação, como o ROS e o Rviz. Recorrendo a estas ferramentas, foi adicionada a restrição da inclinação do terreno ao algoritmo.

Posto isto, com o planeador global pronto, construiu-se um planeador local para desvio de obstáculos que inicialmente não se encontravam no mapa de ocupação.

A trajetória gerada era ainda sub seccionada e descrita por curvas paramétricas de Bézier, sendo esta informação utilizada para estimar a energia consumida pelo robô durante um determinado trajeto. A informação sobre a energia consumida foi útil para o planeamento de trajetória até um dos postos de carregamento dispostos pela vinha já que, ajudou a escolher o posto cujo caminho exige o menor custo energético para o robô. Todo este processo foi executado "off-line"para que o robô tenha acesso imediato a uma trajetória em caso de alerta de nível baixo das baterias. Neste contexto foi ainda implementado outro método de complexidade inferior que estima um custo baseado na distância e diferença de altitude.

Falando de resultados em si, concluiu-se que o algoritmo A* revela gerar caminhos mais seguros quando restringido ao centro de gravidade do robô, resultado em caminhos com menos mudanças de direções e com orientações favoráveis ao robô onde o seu centro de massa não ultrapassa os limites de segurança. Esta restrição não afeta de forma significativa o tempo de processamento nem a distância do caminho gerado mantendo estes parâmetros semelhantes ao algoritmo A* com restrição da orientação. Porém exige um maior tempo de pré-processamento, algo que só precisa de ser feito uma vez.

O planeador local replaneia o caminho utilizando o mesmo algoritmo, sendo que por vezes não encontra um caminho por não ser possível contornar o obstáculo em segurança. O tempo de processamento desta tarefa pode até ser superior ao tempo de gerar um caminho completo porque é aplicado um processo de expansão ao obstáculo local antes de ser feito o replaneamento.

O planeamento "off-line"para postos de carregamento é um processo demorado, porém apenas é executado uma vez por cada mapa e diferente tipo de robô. Foram testados dois métodos para a escolha do posto de carregamento mais adequado:

- estimar a energia consumida pelo robô contabilizando parâmetros como: a trajetória gerada parametrizada com curvas de Bézier, a velocidade, a aceleração, o peso e a inclinação do terreno.
- estimar um custo baseado na distância da trajetória gerada e nas diferenças de altitude entre os vários pontos, para que o posto escolhido não seja necessariamente o que se encontra a uma distância menor.

Os dois métodos geraram resultados semelhantes, no entanto não é possível garantir a mesma precisão que o primeiro método quando se aplicar a grande escala.

6.1 Satisfação dos Objetivos

Todos os objetivos proposto para esta dissertação podem ser considerados cumpridos com uma exceção:

- O algoritmo A* foi adaptado com sucesso para o problema desta dissertação
- Este algoritmo foi dotado da capacidade de replanear o caminho gerado na presença de obstáculos locais
- O algoritmo desenvolvido foi utilizado para o planeamento de trajetórias até aos postos de carregamento
- O algoritmo não foi testado num robô em ambiente simulado ou real

Tem-se algum trabalho futuro que será descrito em seguida.

6.2 Trabalho Futuro

Como possíveis melhorias e trabalho futuro para continuar o realizado no âmbito desta dissertação, podem-se considerar as seguintes abordagens:

- Testar e validar o algoritmo A* com restrições de orientação e centro de gravidade em ambiente simulado e em ambiente real
- Construir o mapa local para replaneamento de trajetória com recurso ao laser do robô
- Adaptar a trajetória parametrizada para o controlo preciso do robô
- Acrescentar ao algoritmo a capacidade de evitar caminhos de grande probabilidade de derrapagem

Anexo A

A.1 Projeção do centro de gravidade do robô (a azul) num plano inclinado

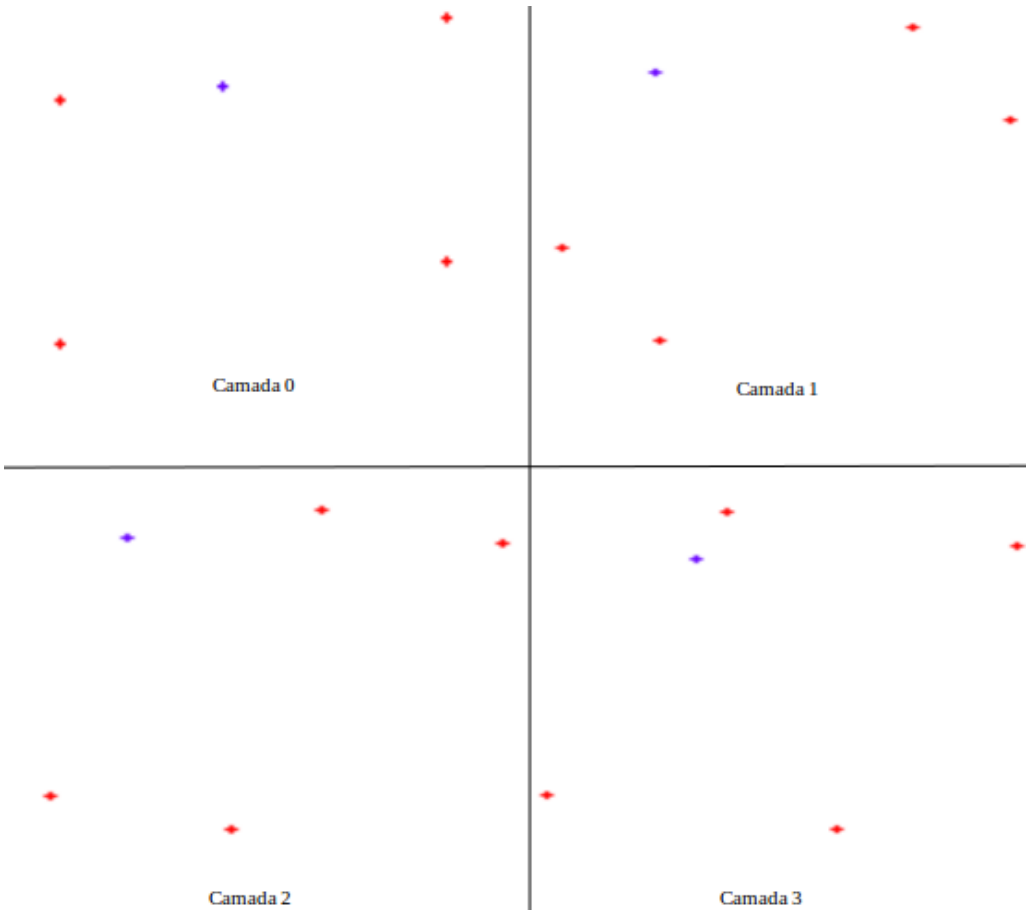

Figura A.1: Projeção nas camadas de 0 a 3

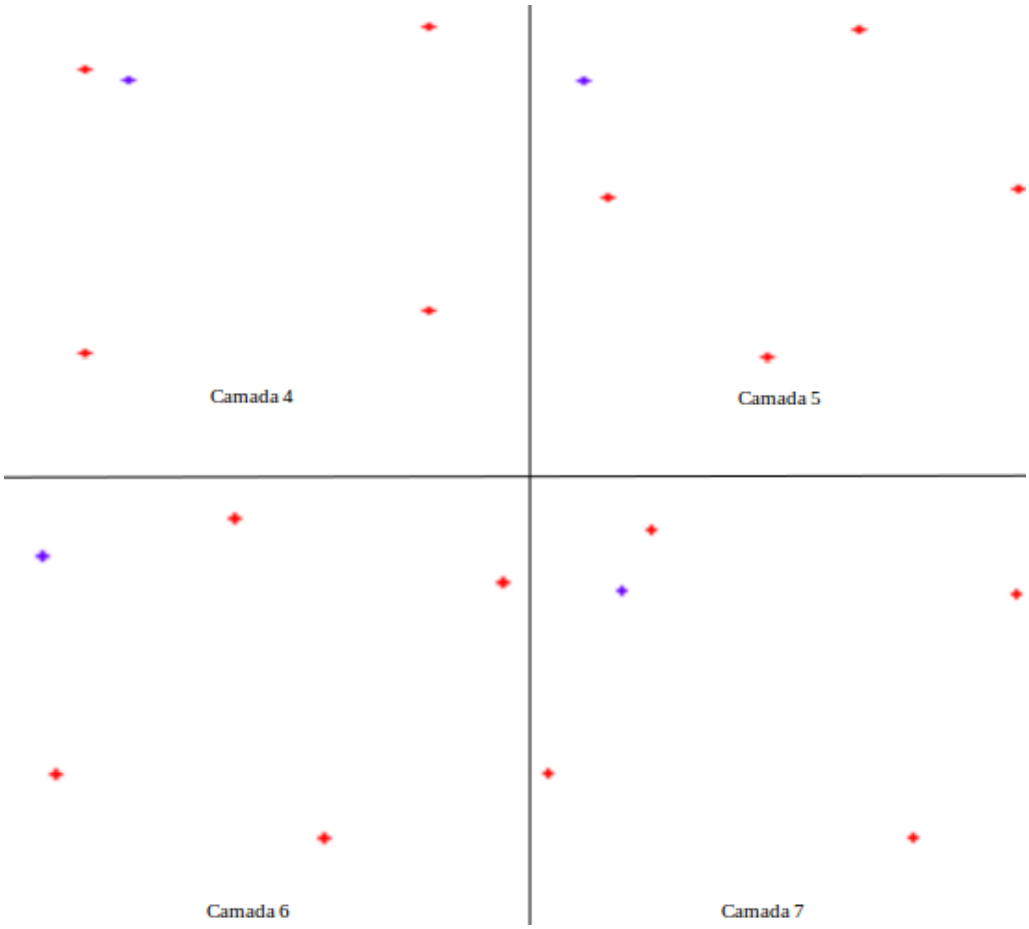

Figura A.2: Projeção nas camadas de 4 a 7

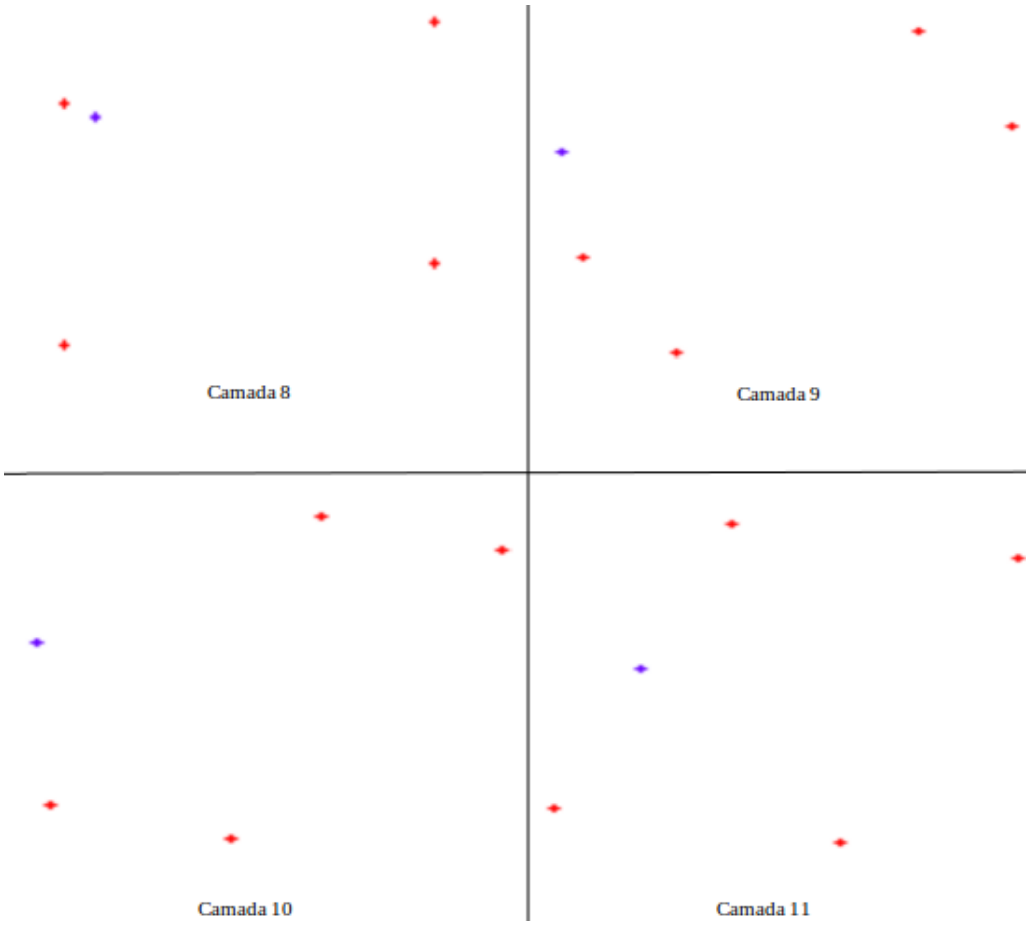

Figura A.3: Projeção nas camadas de 8 a 11

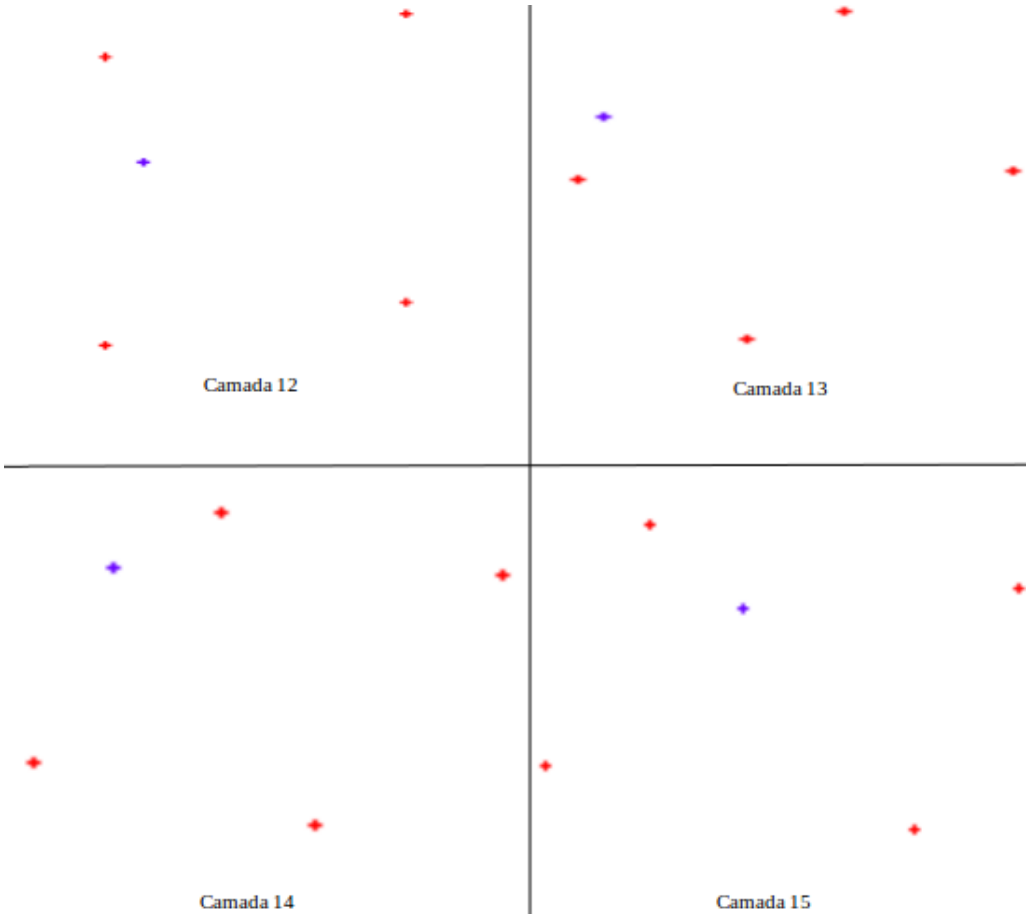

Figura A.4: Projeção nas camadas de 12 a 15

A.2 Projeção do centro de gravidade do robô (a azul) num plano nivelado

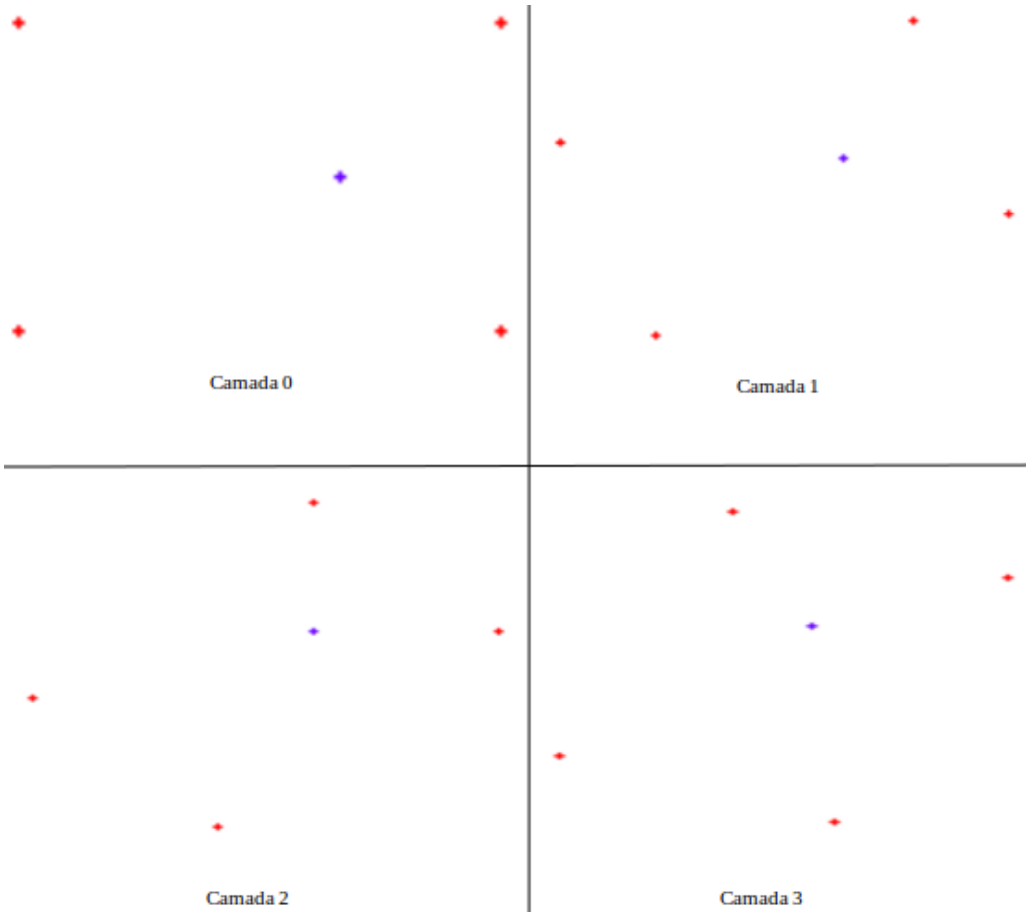

Figura A.5: Projeção nas camadas de 0 a 3

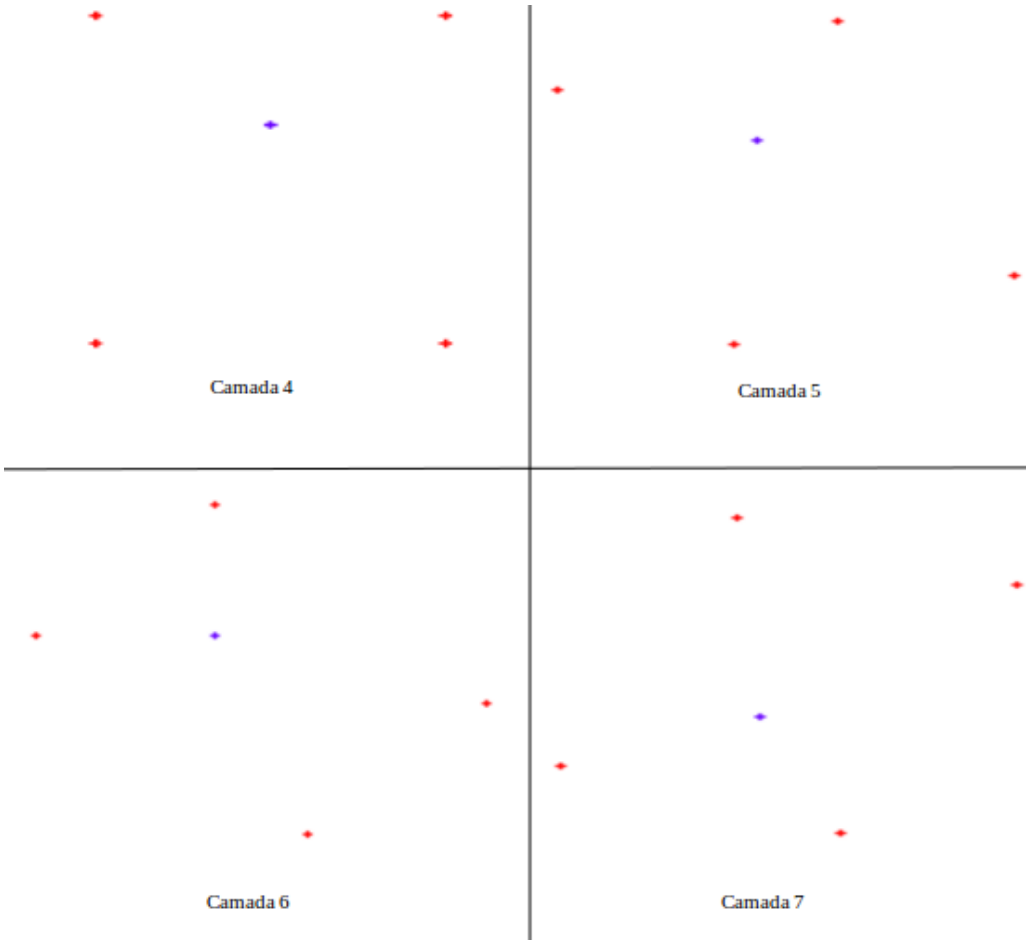

Figura A.6: Projeção nas camadas de 4 a 7

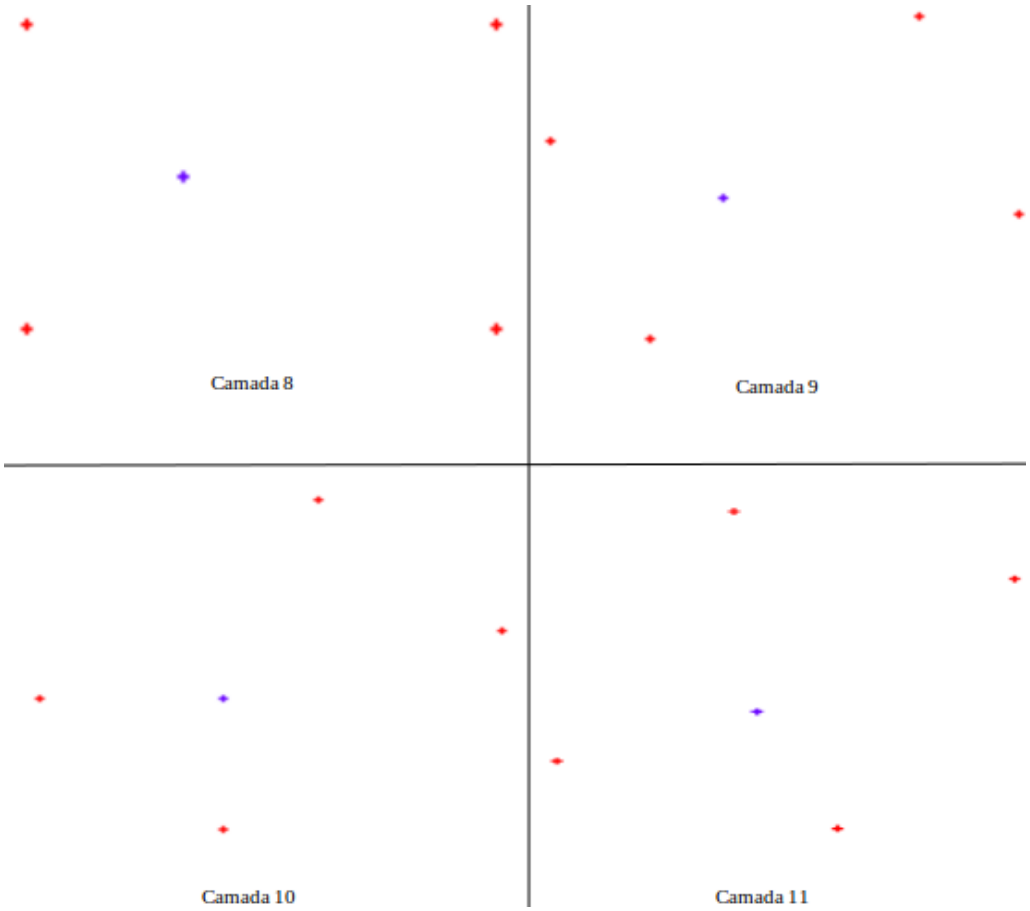

Figura A.7: Projeção nas camadas de 8 a 11

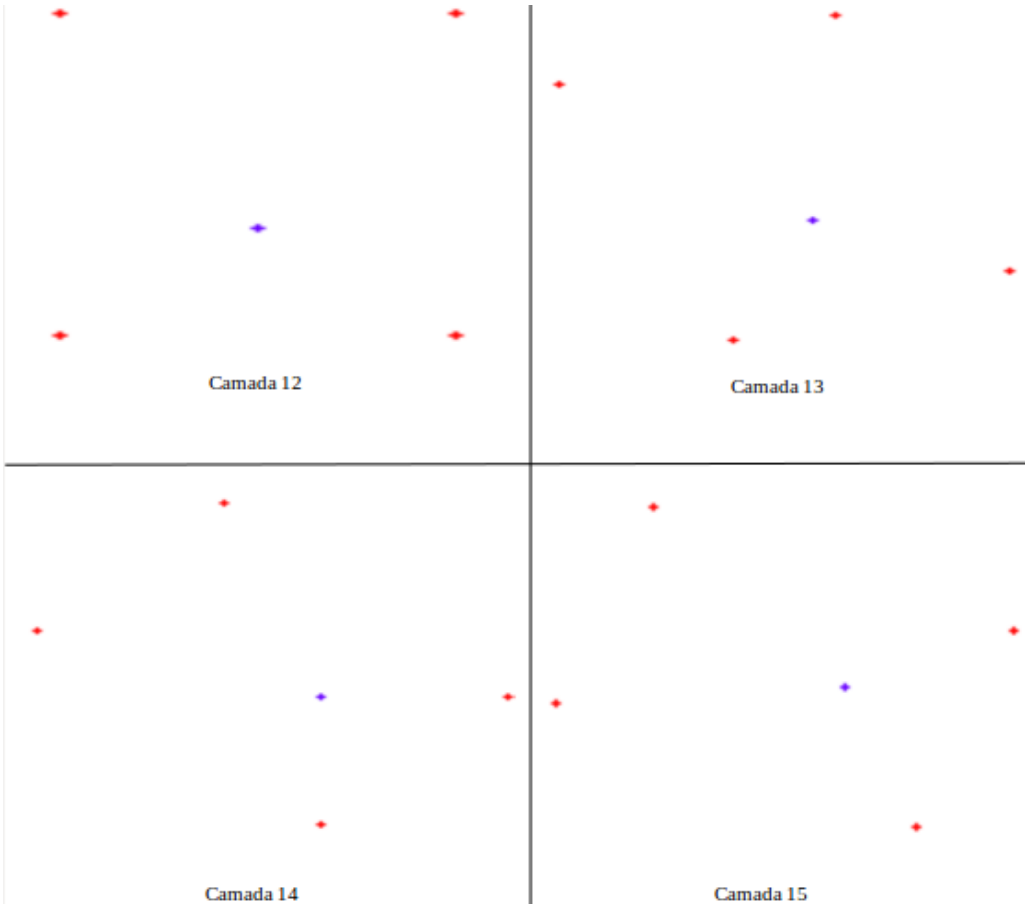

Figura A.8: Projeção nas camadas de 12 a 15

A.3 Mapa de ocupação em camadas com inclinação

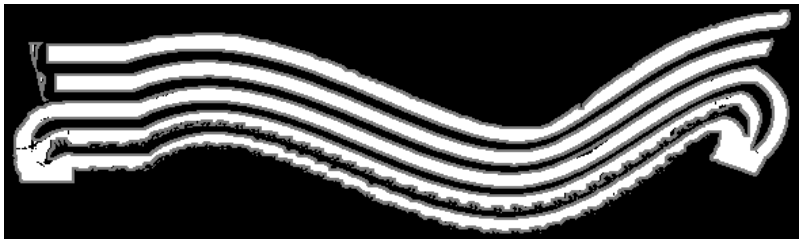

Figura A.9: Mapa de ocupação - camada 0

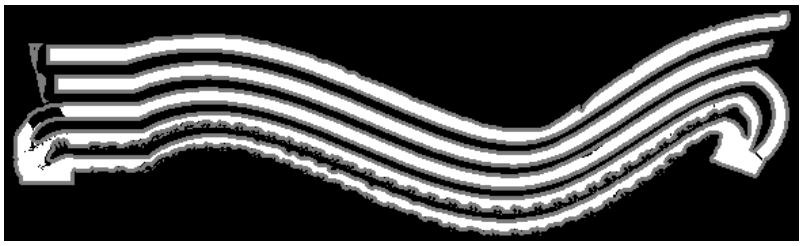

Figura A.10: Mapa de ocupação - camada 1

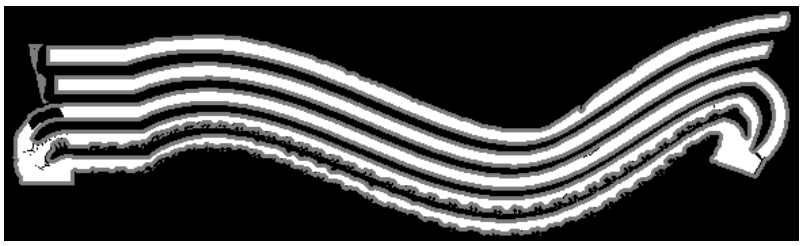

Figura A.11: Mapa de ocupação - camada 2

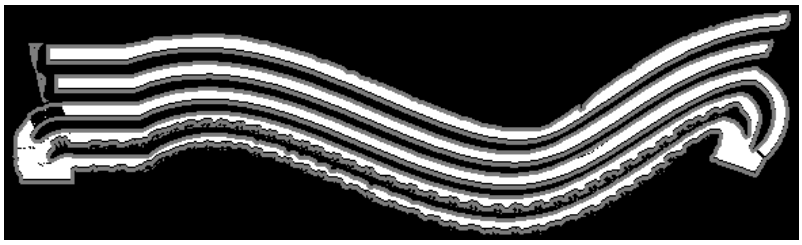

Figura A.12: Mapa de ocupação - camada 3

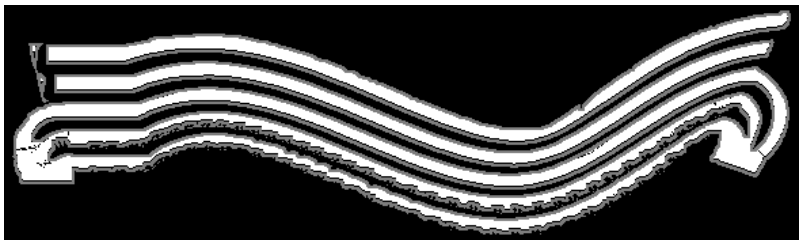

Figura A.13: Mapa de ocupação - camada 4

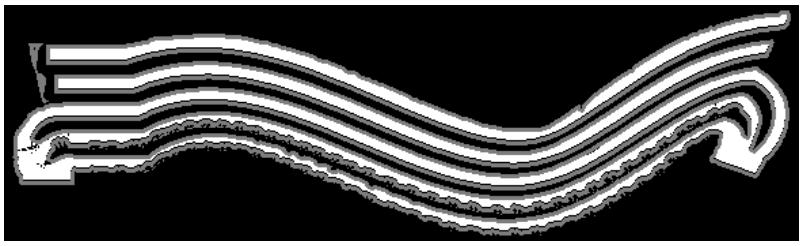

Figura A.14: Mapa de ocupação - camada 5

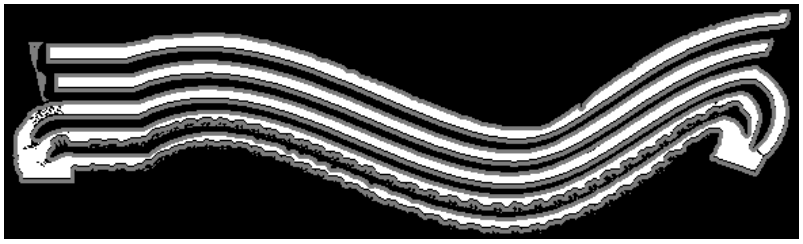

Figura A.15: Mapa de ocupação - camada 6

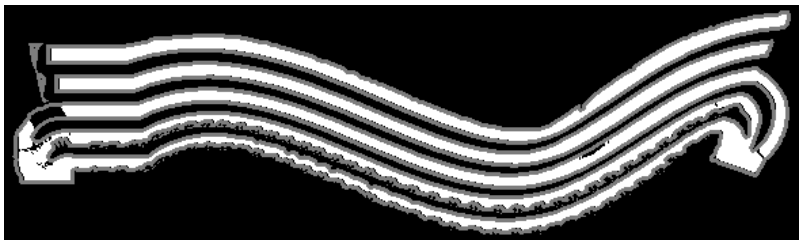

Figura A.16: Mapa de ocupação - camada 7

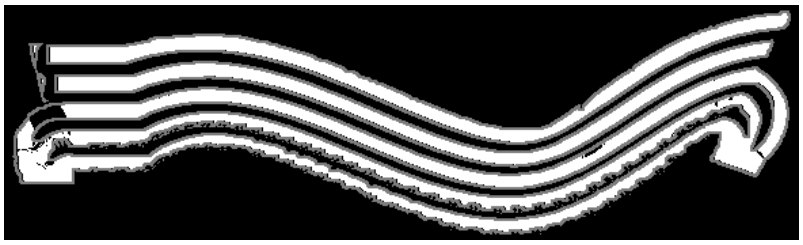

Figura A.17: Mapa de ocupação - camada 8

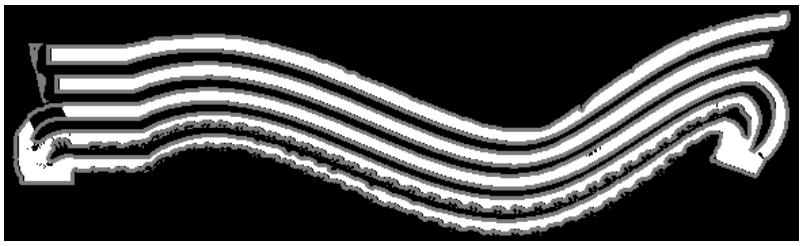

Figura A.18: Mapa de ocupação - camada 9

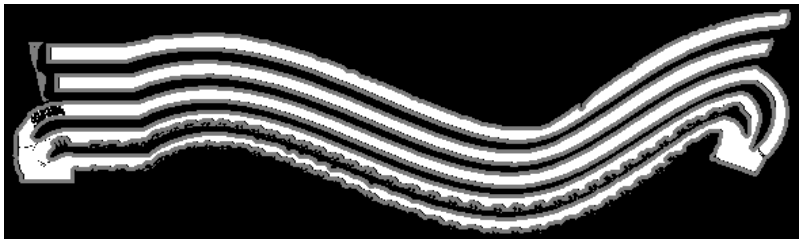

Figura A.19: Mapa de ocupação - camada 10

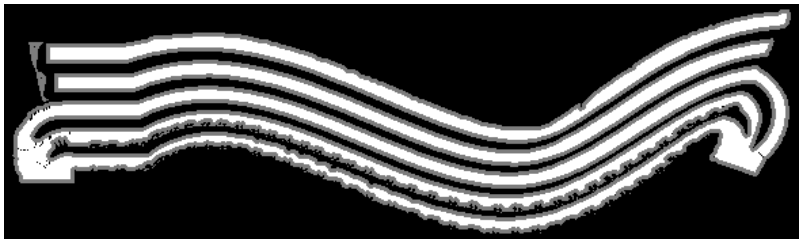

Figura A.20: Mapa de ocupação - camada 11

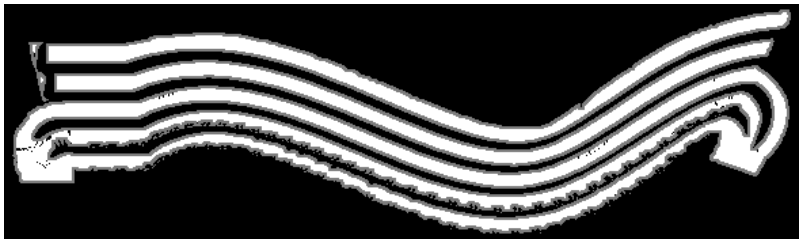

Figura A.21: Mapa de ocupação - camada 12

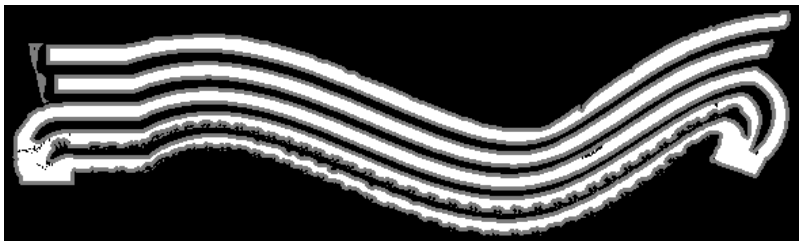

Figura A.22: Mapa de ocupação - camada 13

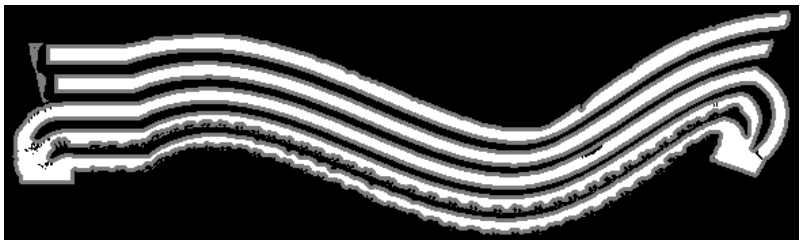

Figura A.23: Mapa de ocupação - camada 14

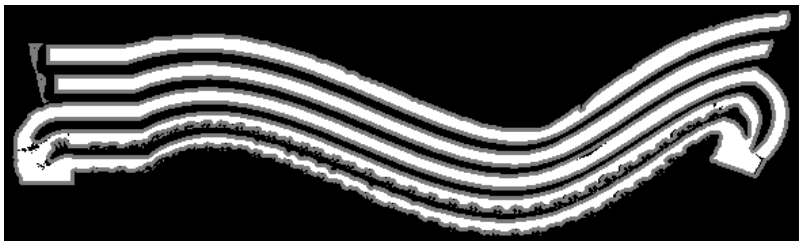

Figura A.24: Mapa de ocupação - camada 15

Anexo B

B.1 Código em Matlab

B.1.1 Criação de curvas de Bézier e cálculo da energia consumida

```
clc; clear all; close all;

path = 'path_file.csv';

data = csvread(path);


t = data(:,1);
%
x = data (:,2);
%
y = data (:,3);
%
Theta = data (:,4);
%
z = data(:,5);
%
yaw = data(:,6);
%
pitch = data(:,7);
%
roll = data(:,8);
%
gx = data(:,9);
%
gy = data(:,10);
%
gz = data(:,11);
%
energy = data(:,12);


```

```
33 ener =0;
34 line = 0;
35
36
37 P1x = x(1);
38 P1y = y(1);
39 P1z = z(1);
40 P1t = t(1);
41 figure
42
43
44 for n = 1:size(Theta)-1
45     w1x = x(n);
46     w1y = y(n);
47     w1z = z(n);
48     w1t= Theta(n);
49     w2x = x(n+1);
50     w2y = y(n+1);
51     w2z = z(n+1);
52     w2t= Theta(n+1);
53
54     sum =0;
55
56     T=t(n+1) - t(n);
57
58     ti= 0:0.01:T;
59
60     u=(ti/T);
61
62
63
64     ug= 0.3*9.8; %atrito e vel
65     %m = (w2y - w1y)/(w2x -w1x) ;
66
67     s= size(x);
68
69
70
71     if ( true)
72
73
74
75         x1= w1x; %x1
76         y1= w1y; %y1
77
78         x2= w1x + cos(w1t)*5; %x2
79         y2= w1y + sin(w1t)*5; %y2
80
81
```

```
82          m = ( y2 -  y1 ) / ( x2 -  x1);
83
84          xa=0;
85          if abs(m) >100000
86           m =inf;
87           xa= x1;
88          elseif abs(m) < 0.001
89               m=0;
90          end
91
92          x1 = w2x;
93          y1 = w2y;
94          z1 = w2z;
95          x2 = w2x + cos(w2t)*5;
96          y2 = w2y + sin(w2t)*5;
97          z2 = w2z;
98
99          m1 = ( y2 -  y1 ) / ( x2 -  x1);
100
101          if abs(m1) >100000
102             m1 = inf;
103             xa= x1;
104          elseif abs(m1) < 0.001
105              m1=0;
106          end
107
108          if m1 ~= inf && m~= inf && m1*m ~= 0
109             xa = (w1x*m - w1y - w2x*m1+w2y)/(m-m1);
110          end
111
112          ya=0;
113
114          if m1 == inf
115              ya = m*xa - m*w1x + w1y;
116          elseif m1~=inf && m~=0
117              ya = m1*xa - m1*w2x + w2y;
118          end
119
120
121
122          if m1== 0
123
124              xa=(ya-w1y)/m + w1x;
125          elseif m==0
126              ya= w1y;
127              xa=((ya-w2y))/m1 + w2x;
128          end
129
130          if (w1t == w2t)
```

```
131             xa= (w1x + w2x)/2;
132             ya= (w1y + w2y)/2;
133          end
134
135
136
137       za= abs ((w2z+w1z)/2 );
138
139
140          for k = 1:100000
141
142              dxu = -4*xa*u(k) + 2*xa + 2*u(k)*w1x + 2*u(k)*w2x - 2*w1x;
143              dyu = -4*ya*u(k) + 2*ya + 2*u(k)*w1y + 2*u(k)*w2y - 2*w1y;
144              dzu = -4*za*u(k) + 2*za + 2*u(k)*w1z + 2*u(k)*w2z - 2*w1z;
145
146              if isnan(dxu) == true
147                  break;
148              end
149
150              if abs(dxu) < 0.0001
151                  dxu=0;
152
153              end
154
155              if abs(dyu) < 0.0001
156                  dyu=0;
157
158              end
159
160                if abs(dzu) < 0.0001
161                  dzu=0;
162
163                end
164
165
166
167                      if (dxu^2 +  dyu^2 ==0)
168                      else
169                      u(k+1) = u(k) + sqrt ( (1/(dxu^2 + dyu^2)) )*0.001;
170                      end
171                      if u(k+1) > 1
172                          break
173                      end
174
175
176
177
178
179
```

```
180          end
181
182                  x1b = ((1-u).^2)*w1x + 2*u.*(1-u)*xa + u.^2*w2x;
183                  y1b = ((1-u).^2)*w1y + 2*u.*(1-u)*ya + u.^2*w2y;
184                  z1b = ((1-u).^2)*w1z + 2*u.*(1-u)*za + u.^2*w2z;
185
186                  vel_q = sqrt( diff(x1b).^2 + diff(y1b).^2  )/0.001;
187
188
189                  vq=size(vel_q);
190
191                  for i = 1:1:vq(2)
192
193                  sum =  sum + 0.06*(-gz(n+1))*vel_q(i)*0.001 - ...
                         0.06*gx(n+1)*vel_q(i)*0.001;
194
195                  end
196
197
198
199         hold on
200         plot(x,y,'o')
201         plot (x1b,y1b,'r', 'linewidth', 1.5 )
202
203         legend('Pontos do trajeto','Curva Bezier')
204
205         hold off
206
207
208
209
210      end
211
212
213       sum;
214      ener = ener+sum;
215
216      line = line +1;
217
218      energ(line) = ener;
219      csvwrite('energ.csv',energ);
220
221  end
```

B.1.2 Representações gráficas de dados (altitude, orientação, aceleração gravítica e energia consumida)

```
1
2  clc; clear all; close all;
3
4  path = 'path_file.csv';
5
6  data = csvread(path);
7
8  dataM= csvread('energ.csv');
9
10
11 t = data(:,1);
12 %
13 x = data (:,2);
14 %
15 y = data (:,3);
16 %
17 Theta = data (:,4);
18 %
19 z = data(:,5);
20 %
21 yaw = data(:,6);
22 %
23 pitch = data(:,7);
24 %
25 roll = data(:,8);
26 %
27 gx = data(:,9);
28 %
29 gy = data(:,10);
30 %
31 gz = data(:,11);
32 %
33 energy = data(:,12);
34 %
35 energyM = dataM(1,:);
36
37
38
39
40
41  figure % new figure
42
43  hold on
44  ax1 = subplot(3,1,1); % top subplot
45
46
47  ax2 = subplot(3,1,2); % bottom subplot
48   hold on
49
```

```
%  ax3 = subplot(4,1,3);
%   hold on

% ax4 = subplot(5,1,4);
%hold on
% axis([0 20 0 20])


 ax5 = subplot(3,1,3);


%plot (ax1,t,x,'r');
%plot (ax1,t,y,'g');
plot (ax1,t,z,'b');




plot (ax2,t,gz,'r');
plot (ax2,t,gy,'g');
plot (ax2,t,gx,'b');

% plot (ax3,t,roll*180/pi,'r');
% plot (ax3,t,pitch*180/pi,'g');
% plot (ax3,t,yaw*180/pi,'b');


%plot (ax4,t, energy);


energyM(size(energyM)+1) = energyM(size(energyM));
plot (ax5, t, energyM);

legend([ax1],{'z'});
legend([ax2], {'gz','gy','gx'});
% legend([ax3], {'Roll','Pitch','Yaw'});

ylabel(ax1, 'Altitude (m)');
ylabel(ax2, 'Aceleracao gravitica (m/s^2) ')
% ylabel(ax3, 'Orientacao')
xlabel(ax5, 'Tempo (s)')
%ylabel(ax4, 'Energia (J) c++')
ylabel(ax5, 'Energia (J)')



hold off
```

Referências

[1] P Raja e S Pugazhenthi. Optimal path planning of mobile robots: A review. *International Journal of Physical Sciences*, 7(9):1314–1320, 2012.

[2] Pedro Luís Cerqueira Gomes da Costa et al. Planeamento cooperativo de tarefas e trajectórias em múltiplos robôs. 2012.

[3] Shuzhi Sam Ge e Yun J Cui. Dynamic motion planning for mobile robots using potential field method. *Autonomous robots*, 13(3):207–222, 2002.

[4] Xinyu Tang, Jyh-Ming Lien, NM Amato, et al. An obstacle-based rapidly-exploring random tree. Em *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, páginas 895–900. IEEE, 2006.

[5] David V. Lu. Global planner, 2017. Disponível em http://wiki.ros.org/global_planner/, acedido a última vez em 30 de Maio de 2017.

[6] Pedro Costa, A Paulo Moreira, e Paulo Costa. Real-time path planning using a modified a* algorithm. *9th Conference on Autonomous Robot Systems and Competitions*, páginas 141–146, 2009.

[7] Elisabete Fernandes, Pedro Costa, José Lima, e Germano Veiga. Towards an orientation enhanced astar algorithm for robotic navigation. Em *Industrial Technology (ICIT), 2015 IEEE International Conference on*, páginas 3320–3325. IEEE, 2015.

[8] C Saranya, Manju Unnikrishnan, S Akbar Ali, DS Sheela, e VR Lalithambika. Terrain based d* algorithm for path planning. *IFAC-PapersOnLine*, 49(1):178–182, 2016.

[9] Olga Maria de Sousa Contente, José Nuno Panelas Nunes Lau, José Francisco Monteiro Morgado, e Raul Manuel Pereira Morais dos Santos. Vineyard skeletonization for autonomous robot navigation. Em *Autonomous Robot Systems and Competitions (ICARSC), 2015 IEEE International Conference on*, páginas 50–55. IEEE, 2015.

[10] Joints - sign convention, note=Disponível em , acedido a última vez em 14 de Junho de 2017, year=2017.

[11] Mihail Pivtoraiko, Ross A Knepper, e Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[12] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Relatório técnico, 1998.

[13] Takayuki Goto, Takeshi Kosaka, e Hiroshi Noborio. On the heuristics of a* or a algorithm in its and robot path-planning. Em *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 2, páginas 1159–1166. IEEE, 2003.

[14] Tiago Pereira Do Nascimento, Pedro Costa, Paulo G Costa, António Paulo Moreira, e André Gustavo Scolari Conceição. A set of novel modifications to improve algorithms from the a* family applied in mobile robotics. *Journal of the Brazilian Computer Society*, 19(2):167–179, 2013.

[15] Daniel Campos. Agrob simulator, 2017. Disponível em https://bitbucket.org/dancampos/agrob_simulator/src, acedido a última vez em 30 de Maio de 2017.

[16] Shuang Liu e Dong Sun. Optimal motion planning of a mobile robot with minimum energy consumption. Em *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, páginas 43–48. IEEE, 2011.

[17] Anis Koubaa. Adding relaxed astar global path planner as plugin in ros, 2017. Disponível em http://www.iroboapp.org/index.php?title=Adding_Relaxed_Astar_Global_Path_Planner_As_Plugin_in_ROS, acedido a última vez em 31 de Maio de 2017.

[18] Luís Santos, Nuno Ferraz, File Neves dos Santos, José Lima, Pedro Costa, e Raul Morais. Automatic recharging system for steep-slope vineyard robots. *Robot'2017*, 2017. Apenas submetido.